AFRL-IF-RS-TR-2001-274
Final Technical Report
January 2002

A PORTABLE SYSTEM FOR INTEGRATING INCONSISTENT AND PARTIAL INFORMATION SOURCES (SIMSPORT)

USC/ISI

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. F708/F244

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

AFRL-IF-RS-TR-2001-274 has been reviewed and is approved for publication.

[illegible]

APPROVED: RAYMOND A. LIUZZI
Project Engineer

[illegible]

FOR THE DIRECTOR: MICHAEL L. TALBERT, Major, USAF
Technical Advisor
Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE Jan 02	3. REPORT TYPE AND DATES COVERED Final Sep 97 - Apr 00

4. TITLE AND SUBTITLE A PORTABLE SYSTEM FOR INTEGRATING INCONSISTENT AND PARTIAL INFORMATION SOURCES (SIMSPORT)	5. FUNDING NUMBERS C - F30602-97-2-0352 PE - 62301E PR - IIST TA - 00 WU - 18
6. AUTHOR(S) Yigal Arens	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) USC/ISI 4676 Admiralty Way Marina del Ray, CA 90292	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency 3701 North Fairfax Drive Arlington, VA 22203-1714 AFRL/IFTD 525 Brooks Rd Rome, NY 13441-4505	10. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2001-274
11. SUPPLEMENTARY NOTES AFRL Project Engineer: Raymond A. Liuzzi, IFTD, 315-330-3577	
12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The objective of this effort was to provide support for the integration, enhancement, transition, and evaluation of component technologies produced from the Defense Advanced Research Projects Agency's (DARPA) Intelligent Integration of Information (1*3), Intelligent Collaboration and Visualization (IC&V), and Information Management (IM) programs. This effort investigated and developed techniques to extend the capabilities of SIMS to integrate data that is incorrect, inconsistent, and/or missing. The SIMS approach was extended to deal with HTML documents accessible through the HTTP protocol. Data mining techniques were used to create automated and semi-automated modeling facilities. A facility for searching for patterns present in data and in the database structure of new sources was created. Overall, this effort has made the integration of multiple, distributed, inconsistent information sources a practical reality and easily portable at reasonable cost.

14. SUBJECT TERMS Computers, software, knowledge bases, data bases, data mining	15. NUMBER OF PAGES 92
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

Table of Contents

1 INTRODUCTION

The Portable System for Integrating Inconsistent and Partial Information Sources (SIMSPORT) project at USC/ISI had two related components. First was the development of **CSIMS**, a C++ version of the SIMS system (which was developed in collaboration with the DARPA-funded BADDInfo project at ISI). The development of CSIMS was a major milestone on the road towards making the software developed in the course of many years of SIMS (Single Interface to Multiple Sources) research at ISI available to a broader audience, which is not accustomed to the use of the LISP programming language. Second, the SIMSPort project continued and extended research on information integration from multiple, heterogeneous sources. The two major extensions were integrating Web sites as information sources, an developing techniques for automating the process of modeling relational databases within the SIMS system—a necessary component of providing the kind of integration capabilities the SIMS system is designed for.

Following a brief background description that will put the current work in context, this report will therefore be divided into two parts: A description of the research effort that was conducted under this cooperative agreement, and description of the CSIMS port.

The CSIMS manual is attached to the report, as further documentation of the work.

1.1 The SIMS Group at ISI

For several years now, with DARPA support, USC/ISI has had a substantial research and prototype development effort in information technology with a particular emphasis on enabling the integration of multiple, distributed heterogeneous sources of data and information. The ISI projects devoted to this pursuit are collectively known as the SIMS group (Single Interface to Multiple Sources).

The SIMS group has developed a modeling system and methodology with which it is possible to declaratively describe the contents, structure and processing capabilities of a variety of types of information sources. We have also developed the SIMS query-processing engine—a central query mediator that handles requests for data possibly distributed among multiple sources while insulating the user (or calling system) from the details of database organization, query language, etc. The SIMS mediator accepts queries formulated against a high-level view of the application domain about which data is stored in the multiple sources. It uses the descriptions of the different sources to construct a plan for obtaining the information requested. This query-plan contains steps that involve constructing subqueries, sending such subqueries to appropriate sources, manipulating the results, performing joins, and so forth.

Noteworthy features of the SIMS approach and system are:

- SIMS addresses the "global integration problem"
    - Information sources are mapped into one domain model
    - Information sources are modeled independently of each other
    - The resulting system is highly extensible and maintainable
- SIMS supports flexible and extensible query languages
    - SQL, SQL with path extensions, the Loom KR language
- SIMS dynamically builds query access plans

- It uses the source descriptions to select relevant information sources and reformulate queries that cannot be answered directly as given
- It minimizes data movement over the network and maximizes parallelism
- It interleaves planning and execution for added flexibility
- It optimizes queries using semantic knowledge
- It exploits knowledge of both the domain and the information sources

- SIMS handles a variety of information sources
    - Relational databases, object-relational databases
    - Web pages and sites
    - Programs with a simple input/output interface

Additional information about SIMS and the efforts descended from it, including downloadable versions of papers describing all aspects of the system as it exists today, can be obtained at http://www.isi.edu/info-agents/. For the status of SIMS at the start of the SIMSPort project, see [Arens et al 96] [Arens et al 94] [Arens et al 93] [Knoblock 95].

1.2 Summary of Project Research Goals

The research performed by the SIMS project has done much to enhance the state of the art in integration of information sources. For a variety of types of information sources, SIMS can now provide single point access to multiple sources, handling queries independent of the distribution of data over sources, of source query languages, data models and organization. SIMS has been demonstrated in the domains of military logistics, battlefield trauma care, and transportation planning. The fact that SIMS mediators are based on solid and principled internal representations of the domain and the sources – and therefore are general and extensible – has been demonstrated by our ability to share these models with similar projects developed elsewhere.

The work being reported on here extended the advances provided by SIMS by developing innovative solutions to critical problems that arise in the actual use:

- A growing amount of information is today stored in HTML documents on the Web.

The SIMS approach was extended to deal with HTML documents accessible through the HTTP protocol. This required that more emphasis be placed on simplifying the process of building "wrappers" for information sources, since a web site typically contains less information that a database. If building wrappers for web sites is too expensive, obtaining information from them will not be cost effective.

- Databases are rife with incorrect, inconsistent, and unfilled fields of data. Source integration must be made robust in the face of these database "defects".

Our system has available to it *domain* and *source models* – a global description of entities and relationships in the domain about which information is stored in the sources, and specific descriptions of the data contained in each source. We used these models – together with SIMS' reasoning engine – to identify cases where data may be present in, or can be derived from, more than one source.

- The system's models of available sources must be accurate and complete. If constructing such models is too difficult and costly, the system will not be used.

We used *data mining* techniques to create automated and semi-automated modeling facilities. We developed a facility for searching for patterns present in the data and in the database structure of new sources. Such patterns provide initial modeling information, and suggest hypotheses that can guide the process of obtaining domain information from human experts.

Taken together, our goal in this work was to make the integration of multiple, distributed, inconsistent information sources a practical reality, easily portable at reasonable cost.

2 SIMSPORT RESEARCH: WEB SITES AS INFORMATION SOURCES*

2.1 Overview

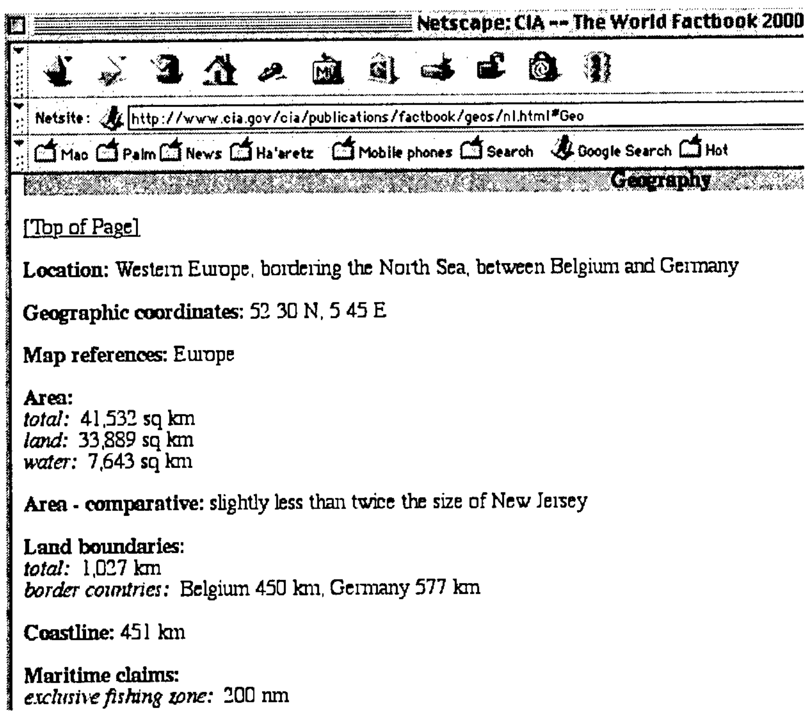
The amount of data accessible via the Web and intranets is staggeringly large and growing rapidly. However, the Web's browsing paradigm does not support many information management tasks. For instance, the only way to integrate data from multiple sites is to build specialized applications by hand. These applications are time-consuming and costly to build, and difficult to maintain.

This section of the report describes Ariadne, an extension of SIMS to a system for extracting and integrating data from semi-structured web sources. Ariadne enables users to rapidly create "information agents" for the Web. Using Ariadne's modeling tools, an application developer starts with a set of web sources—semi-structured HTML pages, which may be located at multiple web sites—and creates a unified view of these sources. Once the modeling process is complete, an end user (who might be the application developer himself) can issue database-like queries as if the information were stored in a single large database. Ariadne's query planner decomposes these queries into a series of simpler queries, each of which can be answered using a single HTML page, and then combines the responses to create an answer to the original query.

The modeling process enables users to integrate information from multiple web sites by providing a clean, well-understood representational foundation. Treating each web page as a relational information source—as if each web page was a little database—provides a simple, uniform representation that makes query planning straightforward. The representation is not very expressive, but we compensate for that by developing intelligent modeling tools that help application developers map complex web sources into this representation.

We will illustrate Ariadne by considering an example application that involves answering queries about the world's countries. An excellent source of data is the CIA World Factbook, which has an HTML page for each country describing that country's geography, economy, government, etc. (See a portion of the entry for the Netherlands on the next page.) (All the web sources in our examples are based on real sources that Ariadne handles, but we have simplified some of them here for expository purposes.)

* This section is based on text written by Craig A. Knoblock, Steven Minton, Jose Luis Ambite, Naveen Ashish, Pragnesh Jay Modi, Ion Muslea, Andrew G. Philpot, and Sheila Tejada, some of whom were supported in part by USC's Integrated Media Systems Center (IMSC) and by a separate U.S. Air Force contract.


Netscape: CIA -- The World Factbook 2000
Netsite: http://www.cia.gov/cia/publications/factbook/geos/nl.html#Geo
Mac Palm News Ha'aretz Mobile phones Search Google Search Hot
Geography

[Top of Page]

Location: Western Europe, bordering the North Sea, between Belgium and Germany

Geographic coordinates: 52 30 N, 5 45 E

Map references: Europe

Area:
total: 41,532 sq km
land: 33,889 sq km
water: 7,643 sq km

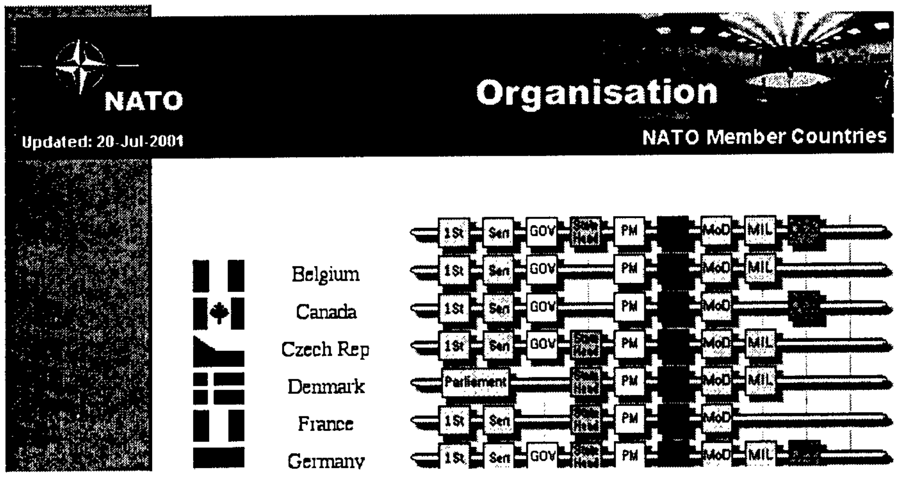
Area - comparative: slightly less than twice the size of New Jersey

Land boundaries:
total: 1,027 km
border countries: Belgium 450 km, Germany 577 km

Coastline: 451 km

Maritime claims:
exclusive fishing zone: 200 nm

Some of the many other relevant sites include the NATO site, which lists the NATO member countries, as shown below, and the World Governments site, which lists the head of state and other government officers for each country (not shown here).



Consider queries such as "What NATO countries have populations less than 10 million?" and "List the heads of state of all the countries in the Middle East". Since these queries span multiple countries and require combining information from multiple sources, answering them by hand is time consuming. Ariadne allows us to rapidly put together a new application that can answer a wide range of queries by extracting and integrating data from prespecified web sources.

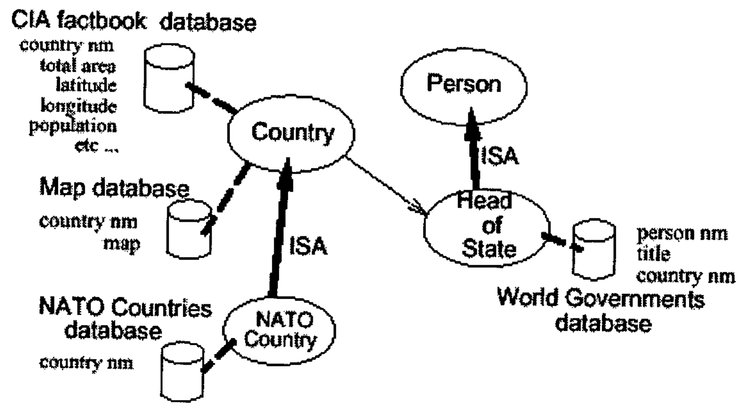
In the following section we describe our basic approach to query planning, where a unifying domain model is used to tie together multiple information sources. We then describe the details of our modeling approach: how we represent and query individual web pages, how we represent the relationships among multiple pages in a single site, and how we integrate data that spans multiple sites. In each section, we also describe the AI methods that are used in modeling and query processing, and how the uniform representational scheme supports these methods.

2.2 Approach to Information Integration

Ariadne's approach to information integration is based heavily on the original SIMS mediator architecture (Arens et al. 1996; Knoblock 1995). The framework consists of two parts: 1) a query planner/executor that determines how to efficiently process a query given the set of available information sources and 2) wrappers that provide uniform access to the information sources so that they can be queried as if they were SQL databases. The original SIMS framework was designed primarily with databases and knowledge bases (and to some extent programs) in mind, but as we will explain, the approach can be extended to handle web sources.

One of the most important ideas underlying SIMS is that for each application there is a unifying domain model that provides a single ontology for the application. The domain model is represented using the Loom knowledge representation system (MacGregor 1988) and is used to describe the contents of each information source. Given a query in terms of the domain model, the system dynamically selects an appropriate set of sources and then generates a plan to efficiently produce the requested data.

To illustrate this, let us first suppose that the information in the three web sites described earlier, the CIA World Factbook, the World Governments site, and the NATO members page, are each available in three separate databases, along with a fourth database containing a map for each country. To define a new information agent, one would first define a domain model that contains the set of terms that the user might want to query about. An example domain model is shown in the figure below. The model contains four classes with some relations between them, e.g., 'NATO Country' is a subclass of 'Country', and 'Country' has a relation called 'Head-of-State', which points to a class with the same name. We then use the domain model to describe each of the individual information sources. This provides the glue for answering queries that span multiple sources. For example, the figure shows that the CIA Factbook is a source for information about Countries, and the World Governments database is a source for Heads of State. Each class has a set of attributes (e.g., total area, latitude, population, etc.) that may be available from one or more sources.



2.3 Query Processing

Queries are presented to the system in terms of the domain model. For example, a query might be "List the heads of state of all the countries whose population is less than ten million." (In actuality, queries are phrased in the Loom KR language, the same language used to express the domain theory. We use English translations for clarity.) The system then decomposes the query into subqueries on the individual sources, such as the World Governments and Factbook sources, producing a partially-ordered query plan consisting of a series of relational operators, i.e., joins, selects, projects, remote subqueries, etc.

The SIMS query planner (Knoblock 1995) was designed primarily for database applications, but database applications typically involve only a small number of databases, while web applications can involve accessing many more sources. The original SIMS planner, which did not scale well to large numbers of sources, was therefore replaced with an approach capable of efficiently constructing very large query plans. We addressed this problem by combining preprocessing techniques with a local-search method for query planning.

In Ariadne, query processing is broken into a preprocessing phase and a query planning phase. In the first phase the system determines the possible ways of combining the available sources to answer a query. Since sources may be overlapping (i.e., an attribute may be available from several sources) or replicated, the system must determine an appropriate combination of sources that can answer the query. The Ariadne source selection algorithm (Ambite et al. 1998) preprocesses the domain model so that the system can efficiently and dynamically select sources based on the classes and attributes mentioned in the query.

In the second phase, Ariadne generates a plan using a method called Planning-by-Rewriting, developed by Ambite and Knoblock (Ambite and Knoblock 1997; 1998). This approach takes an initial, suboptimal plan and then attempts to improve it by applying rewriting rules. In the case of query planning, producing an initial, suboptimal plan is straightforward; we can generate an initial plan in O(n) time, where n is the length of the query, based on a depth-first parse of the query. The rewriting process iteratively improves the query via a local search process that can change both the sources used to answer a query and the order of the operations on the data.

Consider the processing required to retrieve all NATO countries that have a population of less than 10 million. Using the domain model shown earlier, the source selection step would determine that the NATO source is the only source for the class of NATO countries. This source provides only the names of the NATO countries and not their populations. However, the population information can be extracted from the Factbook source since it provides data for a superclass of NATO countries. The reasoning to combine sources in this way is done efficiently by precomputing the way sources can be combined before any queries are processed.

The next step in the example is to construct a plan for efficiently retrieving and combining the data. In this case, the system might first construct an initial plan that retrieves the data from the NATO source, separately retrieves the names and population for all countries from the Factbook source, and then combines the data locally, which is very costly since the Factbook source is quite large. This initial, suboptimal plan is then improved in a series of rewriting steps that would order the retrieval of the NATO source before the Factbook source so that only population data on the NATO countries would need to be retrieved. The optimized plan would then be executed, returning only Denmark, which has a population of just over 5 million.

Because Ariadne combines an efficient source selection algorithm with an efficient, anytime planning algorithm, the system can produce query plans for web environments in a robust, efficient manner. Ariadne's development was aided by the fact that the relational algebra is very simple and well understood, so that we could concentrate on the issues involved in searching for a plan, rather than on the underlying plan representation, which simply consists of a partially ordered set of relational operators. To move to the Web, we only needed one extension to the basic representation, which was the inclusion of "binding patterns" (Kwok and Weld 1996). That is, unlike database sources, web sources may have input/output constraints (e.g., a stock quote server requires a ticker symbol in order to retrieve a stock quote). This is a small extension that is naturally handled by the source selection algorithm and planning operators.

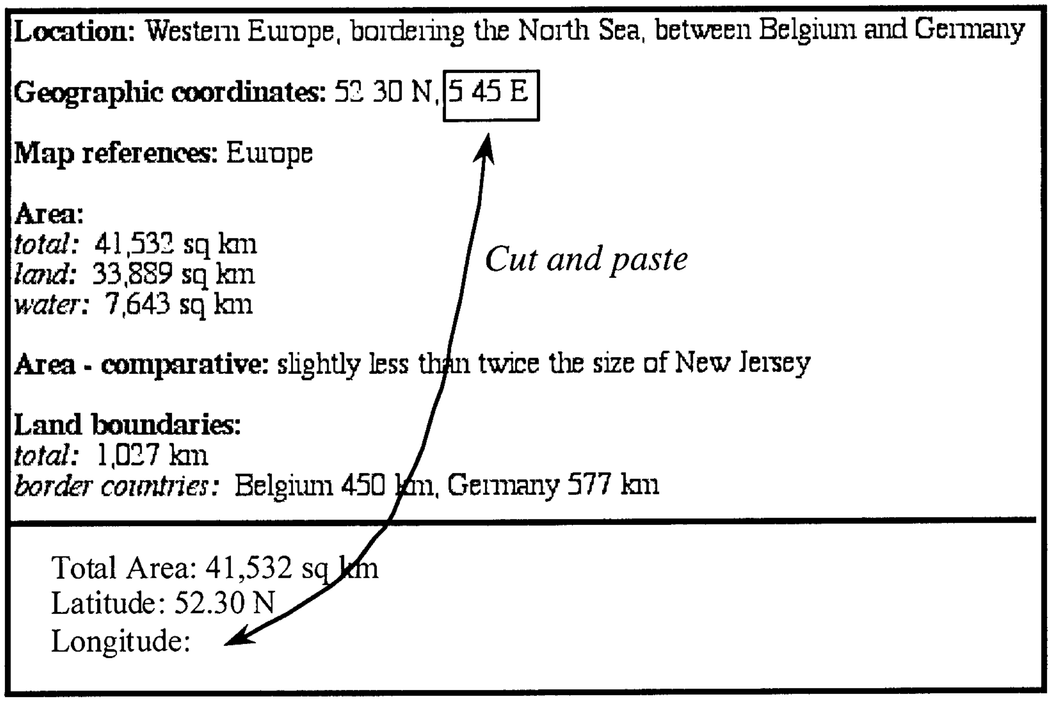
In the remainder of the section of the report we consider in more detail the modeling issues involved in creating a database-like view of the Web.

2.4 Modeling the Information on a Page

The previous section described how the planner decomposes a complex query into simple queries on individual information sources. To treat a web page as an information source so that it can be queried, Ariadne needs a wrapper that can extract and return the requested information from that type of page. While we cannot currently create such wrappers for unrestricted natural language texts, many information sources on the Web are semistructured. A web page is semistructured if information on the page can be located using a concise formal grammar, such as a context-free grammar. Given such a grammar, the information can be extracted from the source without recourse to sophisticated natural language understanding techniques. For example, a wrapper for pages in the CIA Factbook would be able to extract fields such as the Total Area, Population, etc. based on a simple grammar describing the structure of Factbook pages.

Our goal is to enable application developers to easily create their own wrappers for web-based information sources. To construct a wrapper, we need both a semantic model of the source that describes the fields available on that type of page and a syntactic model, or grammar, that describes the page format, so the fields can be extracted. Requiring developers to describe the syntactic structure of a web page by writing a grammar by hand is too demanding, since we want to make it easy for relatively unsophisticated users to develop applications. Instead, Ariadne has a "demonstration-oriented user interface" (DoUI) where users show the system what information to extract from example pages. Underlying the interface is a machine learning system for inducing grammar rules.

The figure on the next page shows how an application developer uses the interface to teach the system about CIA Factbook pages, producing both a semantic model and a syntactic model of the source. The screen is divided into two parts. The upper half shows an example document, in this case the Netherlands page. The lower half shows a semantic model, which the user is in the midst of constructing for this page. The semantic model in the figure indicates that the class Country has attributes such as Total Area, Latitude, Longitude, etc. The user constructs the semantic model incrementally, by typing in each attribute name and then filling in the appropriate value by cutting and pasting the information from the document. In doing so, the user actually accomplishes two functions. First, he provides a name for each attribute. Notice that he can choose the same names as used in the document (e.g., "Total area") or he can choose new/different names (e.g., "Latitude"). As we will explain later, the attribute names have significance, since they are the basis for integrating data across sources.


Location: Western Europe, bordering the North Sea, between Belgium and Germany

Geographic coordinates: 52 30 N, 5 45 E

Map references: Europe

Area:
total: 41,532 sq km
land: 33,889 sq km
water: 7,643 sq km

Area - comparative: slightly less than twice the size of New Jersey

Land boundaries:
total: 1,027 km
border countries: Belgium 450 km, Germany 577 km

Total Area: 41,532 sq km
Latitude: 52.30 N
Longitude:

The second function achieved by the user's demonstration is to provide examples so that the system can induce the syntactic structure of the page. Ideally, after the user has picked out a few examples for each field, the system will induce a grammar sufficient for extracting the required information for all pages of this type. Unfortunately, grammar induction methods may require many examples, depending on the class of grammars being learned. However, we have observed that web pages have common characteristics that we can take advantage of, so that a class of grammars sufficient for extraction purposes can be rapidly learned in practice.

More specifically, we can describe most semistructured web pages as embedded catalogs. A catalog is either a homogeneous list, such as a list of numbers, (1,3,5,7,8), or a heterogeneous tuple, such as a 3tuple consisting of a number, a letter, and a string, (1,A,"test"). An embedded catalog is a catalog where the items themselves can be catalogs. As an example, consider a CIA Factbook page. The top level consists of an 8-tuple distinguished by section headings: Geography, People, etc. The Geography section is a tuple consisting of Map References, Area, Coastline, etc. These can be decomposed further if necessary; Coastline is a tuple consisting of a number and the string "km".

Because web pages are intended to be human readable, special markers often play a role identifying the beginning or ending of an item in an embedded catalog, separating items in a homogeneous list, and so on. These distinguishing markers can be used as landmarks for locating information on a page. For instance, to find the longitude, simply skip down to the heading "Geography", then to "Geographic Coordinates:", and then skip past the first comma.

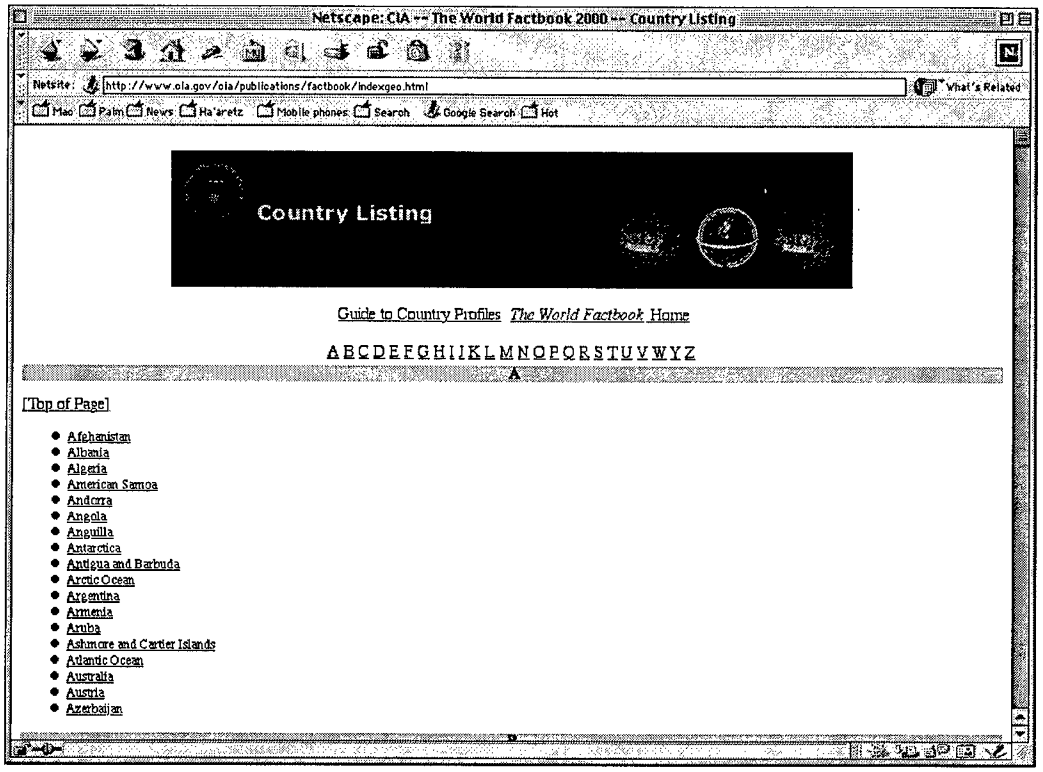
A landmark grammar describes the position of a field via a sequence of landmarks, where each landmark is itself described by a deterministic finite automaton. Some of our work (Muslea et al. 1998) shows that in practice, a subclass of landmark grammars (linear, augmented landmark grammars) can be learned rapidly for a variety of web pages using a greedy covering algorithm. There are several reasons for this. Firstly, because web pages are intended to be human readable, there is often a single landmark that distinguishes or separates each field from its neighbors. Therefore the number of landmarks for a field in an embedded catalog will generally be equal to its "depth" in the catalog. Since most catalogs are very shallow, this means that the length of the grammar rules to be learned will be very small, and learning will be easy in practice. Secondly, during the demonstration process, users traverse a page from top-to-bottom, picking out the positive examples of each field. Any position on the page that is not marked as a positive example is implicitly a negative example. Thus, for every positive example identified by the user, we obtain a huge number of negative examples that the covering algorithm can use to focus its search.

The modeling tool we have described enables unsophisticated users to turn web pages into relational information sources. But it has a second advantage as well. If the format of a web source changes in minor respects, the system could induce a new grammar by reusing examples from the original learning episode, without any human intervention (assuming the underlying content has not changed significantly). This is a capability we are still exploring.

2.5 Modeling the Information in a Site: Connections between Pages

The previous section showed how Ariadne extracts information from a web page to answer a query. However, before extracting information from a page, Ariadne must first locate the page in question. Our approach, described in this section, is to model the information required to "navigate" through a web site, so that the planner can automatically determine how to locate a page.

For example, consider a query to our example information agent asking for the population of the Netherlands. To extract the population from the Factbook's page on the Netherlands, the system must first find the URL for that page. A person faced with the same task would look at the index page for the Factbook, shown (in part) in the figure below, which lists each country by name together with a hypertext link to the page in question. In our approach, Ariadne does essentially the same thing. The index page serves as an information source that provides a URL for each country page. These pages in turn serve as a source for country-



http://www.oia.gov/oia/publications/factbook/indexgeo.html

Country Listing

Guide to Country Profiles *The World Factbook* Home

A B C D E F G H I J K L M N O P Q R S T U V W Y Z

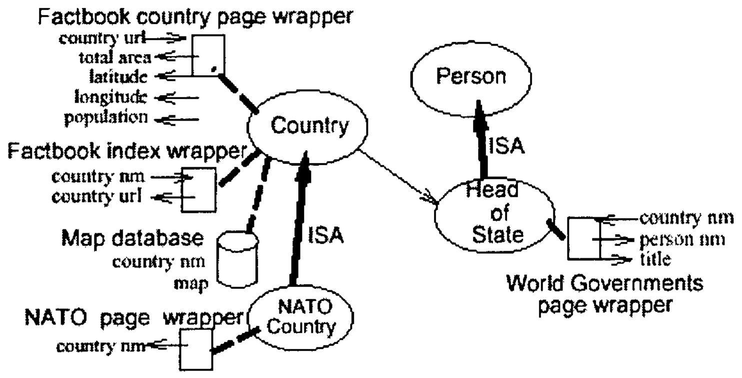
A

[Top of Page]

- Afghanistan
- Albania
- Algeria
- American Samoa
- Andorra
- Angola
- Anguilla
- Antarctica
- Antigua and Barbuda
- Arctic Ocean
- Argentina
- Armenia
- Aruba
- Ashmore and Cartier Islands
- Atlantic Ocean
- Australia
- Austria
- Azerbaijan

specific information.

To create a wrapper for the index page, the developer uses the approach described in the last section, where we illustrated how a wrapper for the Factbook's country pages is created. There is only one difference: this wrapper only wraps a single page, the index page. The developer creates a semantic model indicating that the index page contains a list of countries, each with two attributes, country-nm and country-URL. The learning system induces a grammar for the entire page after the developer shows how the first few lines in the file should be parsed.

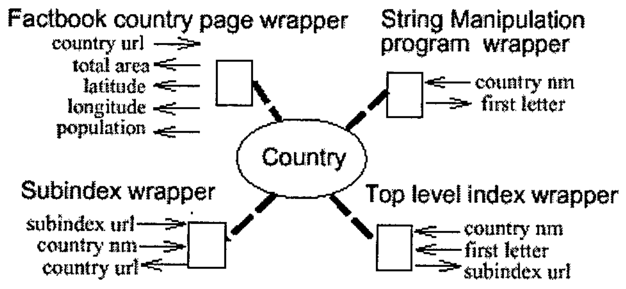


As the wrappers for each source are developed, they are integrated into the unifying domain model. The figure above shows the domain model for the completed geopolitical agent. (Notice that we have substituted web source wrappers for the hypothetical databases used previously.) To create the domain model, the developer specifies the relationship between the wrappers and the domain concepts. For instance, the developer specifies that the Factbook country wrapper and the Factbook index wrapper are both information sources for "country" information, and he identifies which attributes are keys (i.e., unique identifiers). In the example, "country-nm" and "country-URL" are both keys. Binding constraints specify the input and output of each wrapper (shown by the small directional arrows in the model). The country page wrapper takes a country-URL, and acts as a source for "total area", "population", "latitude", etc. The index wrapper takes a country name and acts as a source for "country-URL". Given the domain model and the binding constraints, the system can now construct query plans. For instance, to obtain the population of a country given its name, the planner determines that the system must first use the country name to retrieve the country-URL from the index page wrapper, and then use the country-URL to retrieve the population data from the country page wrapper.

Explicitly modeling 'navigation' pages, such as the Factbook index, as information sources enables us to reuse the same modeling tools and planning methodology underlying the rest of the system. The approach works well in part because there are only two common types of navigation strategies used on the Web—direct indexing and form-based retrieval. We have already seen how index pages are handled; form-based navigation is also straightforward. A wrapper for an HTML form simply mimics the action of the form, taking as input a set of attributes, each associated with a form parameter name, and communicating with the server specified in the form's HTML source.

When the resulting page is returned, the wrapper extracts the relevant attributes in the resulting page. Imagine, for instance, a form-based front end to the Factbook, where the user types in a country name and the form returns the requested country page. To create a wrapper for this front end, the developer would first specify that the parameter associated with the typein box would be filled by a "country-nm". He would then specify how the system should extract information from the page returned by the form using the approach described in the last section.

The Factbook example described in this section illustrates our basic approach to modeling navigation pages. Many web sites are more complex than the Factbook. The approach still works, but the models become more involved. For instance, indexes can be hierarchical, in which case each level of the hierarchy must be modeled as an information source. Imagine the top-level Factbook index was a list of letters, so that clicking on a letter "C" would produce an index page for countries starting with "C" (a "subindex"). We would model this top-level index as a relation between letters and subindex-URL's. To traverse this index, we also need an information source that takes a country name and returns the first letter of the name (e.g., a string manipulation program). Thus, altogether four wrappers would be involved in the navigation process, as shown in the following figure.

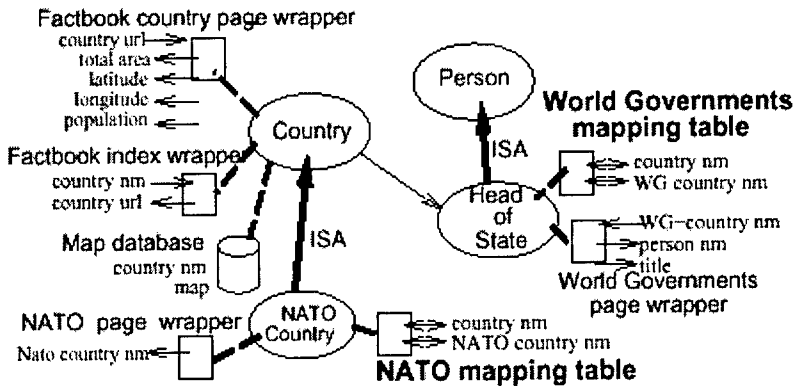


Given a query asking for the Netherlands' population, the first wrapper would take the name "Netherlands", call the string manipulation program, and return the first letter of the name, "N". The second wrapper would take the letter "N", access the top-level index page, and return the subindex URL. The third wrapper would take the subindex-URL and the country name, access the subindex page for countries starting with "N", and return the country-URL. Finally, the last wrapper would take the country-URL and access the Netherlands page. The advantage of our approach is that all these wrappers are treated uniformly as information sources, so the query planner can automatically determine how to compose the query plan. Furthermore, the wrappers can be semi-automatically created via the learning approach described earlier, except for the string manipulation wrapper, which is a common utility.

2.6 Modeling Information Across Sites

Within a single site, entities (e.g., people, places, countries, companies, etc.) are usually named in a consistent fashion. However, across sites, the same entities may be referred to with different names. For example, the CIA Factbook refers to the "Vatican City" while the World Governments site refers to "The Holy See". Sometimes formatting conventions are responsible for differences, such as "Denmark" vs. "Denmark, Kingdom of". To make sense of data that spans multiple sites, we need to be able to recognize and resolve these differences.

Our approach is to select a primary source for an entity's name and then provide a mapping from that source to each of the other sources where a different naming scheme is used. An advantage of the Ariadne architecture is that the mapping itself can be represented as simply another wrapped information source. One way to do this is to create a mapping table, which specifies for each entry in one data source what the equivalent entity is called in another data source. Alternatively, if the mapping is computable, it can be represented by a mapping function, which is a program that converts one form into another form.



The figure above illustrates the role of mapping tables in our geopolitical information agent. The Factbook is the primary source for a country's name. A mapping table maps each Factbook country name into the name used in the World Governments source (i.e., WG-countrynm). The mapping source contains only two attributes, the (Factbook) country name and the WG-country-nm. The NATO source is treated similarly. So, for example, if someone wanted to find the Heads of State of the NATO countries, the query planner would retrieve the NATO country names from the NATO wrapper, map them into (Factbook) country names using the NATO mapping table, then into the World Government country names using the World Governments mapping table, and finally retrieve the appropriate heads of state from the World Governments wrapper.

Currently, mapping tables and functions must be created manually, but we are developing a semiautomatic method for building mapping tables and functions by analyzing the underlying data in advance. The basic idea is to use information retrieval techniques to provide an initial mapping (e.g., (Cohen 1998)), and then use additional data in the sources to resolve any remaining ambiguities via statistical learning methods (e.g., (Huang and Russell 1997)). For example, both the Factbook and the World Governments sources list the title of the Heads of State. (The Factbook lists the name of the Heads of State as well but, unlike the World Governments site, the information is often out of date. This is one reason why the World Governments site is useful.) This information can help determine that the Factbook's "North Korea" and "South Korea" refer respectively to the World Government's "Democratic Republic of Korea" and "Republic of Korea", rather than the other way around. Our approach can also be used to automatically update mapping tables when new sources are released. For instance, each year a new version of the CIA Factbook is released, and sometimes countries have new names, or countries merge or split. These name confusions can often be resolved using geographical information (e.g., land area, latitude and longitude).

2.7 Applications

Below we list some Ariadne applications we are in the process of being developed, illustrating the generality of our approach:

Worldwide Geographic Information Server: We are collaborating with another group that is building a geographic information system that integrates a variety of map-based information sources. These sources include satellite images, detailed street maps, parcel data, historical aerial photographs, etc. We are using Ariadne to extract geographically referenced data from the Web and integrate it with map data. We have built a system using Ariadne that extracts restaurant data from the Zagat Restaurant Reviews site, feeds the restaurant address into a geocoder, and then places the restaurant on an aerial map. Other web sources that we plan to incorporate include census data, US Geological Survey data, and real estate data from the Multiple Listing Service.

Electronic Catalog Access: We are applying Ariadne to provide access to online electronic catalogs for the Defense Logistics Agency. One implemented application provides real-time access to pricing and availability data from the General Services Administration web pages. This application accesses only a single site, but retrieves pricing data for parts by extracting and integrating data from multiple pages in the site.

Financial Information Agent: We have done initial work on an agent that accesses stock quote servers, stock exchange sources, and the SEC's EDGAR Archives (which contains copies of financial filings, such as annual reports, by publicly traded companies and mutual funds). By integrating these sources, the agent could answer queries such as "Find all airline companies whose stock has risen more than thirty percent in the last year" and "Find all people who serve as directors of two or more companies located in Los Angeles". Using the modeling tools described earlier, users could also include their own personal financial data sources, tailoring the system to their needs.

2.8 Summary

There are many examples of impressive AI systems based on relatively simple representational schemes. In the realm of planning, recent examples include SATplan (Kautz and Selman 1996) and Graph-plan (Blum and Furst 1995); the former employs a propositional CSP approach, the latter, a graph-based search. In machine learning, propositional learning schemes (e.g., decision trees) have been dominant. Though it is often difficult to understand exactly what a simple representational scheme buys you computationally, one thing seems clear: systems with simple representations are often easier to design and understand.

We believe that Ariadne is successful, in terms of the broad applicability of the approach, because it combines a simple representation scheme with sophisticated modeling tools that map web information sources into this simple representation. Ariadne capitalizes on a representation scheme adopted from database systems, where the world consists of a set of relations (or tables) over objects, and simple relational operators (retrieve, join, etc.) are composed to answer queries. This representation makes it straightforward to integrate multiple databases using an AI planner. Ariadne's planner can efficiently search for a sequence of joins, selections, etc. that will produce the desired result without needing to do any sophisticated reasoning about the information sources themselves.

The Web environment is much richer than the database world, of course. What makes Ariadne possible are the modeling tools that enable a user to create a database-like view of the Web. Where our approach becomes challenging (and could break down) is in situations where the "natural" way to represent a web source is not possible due to limitations of the underlying representation.

One such limitation is that Ariadne cannot reason about recursive relations. (To do this properly would require query plans to contain loops.) This has many practical ramifications. For example, consider web pages that have a 'more' button at the bottom, such as Alta Vista's response pages. It would be natural to represent each 'more' button as a pointer to the next page in a list, but there is no way to do this without a recursive relation. Instead, we can build knowledge about 'more' buttons in our wrapper generation tools, so the process of following a 'more' buttons is done completely within a wrapper, hiding the complexity from the query planner.

Another ramification of the planner's inability to reason about recursive relations shows up with hierarchical indexes like Yahoo, where there is no fixed depth to the hierarchy. The natural way to model such pages is with a parent-child relation. Instead, the alternative is to build a more sophisticated wrapper that computes the transitive closure of the parent-child relationship, so that we can obtain all of a node's descendants in one step.

There is an obvious tension between the expressiveness of the representation and the burden we place on the modeling tools. Our approach has been to keep the representation and planning process simple, compensating for their weaknesses by relying on smarter modeling tools. As we have described, the advantage is that we can incrementally build a suite of modeling tools that use machine learning, statistical inference, and other AI techniques, producing a system that can handle a surprisingly wide range of tasks.

3 AUTOMATING MODEL CONSTRUCTION FROM DATABASES*

The process of constructing SIMS models of databases is an extremely important one, since the ability of the system to eventually find the correct data depends on the accuracy of the models. But it is also a time consuming and error prone process. With funding from this contract, we developed a prototype system for automating this process for relational databases. The system is called *McKey*, and is described in a paper that was presented at the SPIE Confernece on Data Mining and Knowledge Discovery: Theory, Tools, and Technology, Proceedings of SPIE Vol. 3695, edited by B. V. Dasarathy, 1999.

A conceptual model of a database, such as an Entity-Relationship (ER) model, is a specification of objects, attributes, and their relation-ships. A conceptual model plays important roles in developing successful database applications. Although critical, a conceptual model of a legacy database may not be always available in practice, and discovering and constructing such a model from the data, and from the data only, is a challenging problem. In this paper, we develop a new approach to address object identification and model construction. Our approach has many favorable features, including its robustness in dealing with noise data and scalability to large databases and data sets. We implement this approach in a system called McKey (Model Construction with Key identification) for discovering and building ER models from instances of large legacy databases. We apply McKey to three very large legacy databases, and obtain comprehensive models within hours, which provides a very large savings of manpower.

* This section of the report was written by Wei-Min Shen, Weixiong Zhang, Xuejun Wang, and Yigal Arens.

Introduction

A conceptual model is a specification of objects, attributes, and their relationships. Conceptual models include Entity-Relationship (ER) model [7, 15, 27] used in the database community, frames and semantic nets [6, 11, 18] developed in the artificial intelligence area, and their various extensions. A conceptual database model, such as an ER model which specifies objects and their relationships in a database, plays irreplaceable roles in both understanding a data set and in developing successful database applications using the data.

3.1 Scientific and engineering importance

However, a conceptual model may not always be available for a legacy database due to various reasons, such as long histories of modifications and extensions, poor documentation and maintenance, and turnovers of domain experts. It may be too costly, or even impossible, to abandon the data as well as the information stored in legacy databases since the data was collected and evolved over a long period of time. Therefore, discovering and constructing an accurate as well as comprehensive conceptual model for a legacy database is an important engineering practice.

A conceptual database model is usually needed to understand the logic of the data in the database and the design principle used in the first place, to effectively utilize the data, to reduce maintenance cost [9, 16], etc. In addition to these traditional uses, a conceptual data model is also critical in developing successful database applications, especially in those through information integration. When integrating information from multiple heterogeneous data sources, a common model of an application domain is usually required [2, 3, 13]. Such a domain model can be typically built upon the conceptual models of individual data sources. In addition, a conceptual model of a data source is also needed to wrap the data source for translating queries in different query languages [12, 26].

If it is not automated, however, the model construction process typically requires extensive experience of domain experts and intensive labor of knowledge engineers. Automatic discovery and construction of a conceptual model from a legacy database is, on the other hand, a nontrivial task. This is because the valuable schema information in a legacy database, such as objects and their identities, is usually buried in a large amount of data that are noisy, corrupted, and irregularly structured.

3.2 Previous work

Constructing conceptual models from relational databases is generally referred to as reverse engineering, and there are a lot of published work on this topic. The main concern of reverse engineering is how to construct a conceptual model when information about objects, their keys and relationships are known. For example, [8] and [9] describe an approach for extracting an Extended Entity-Relationship (EER) model from a relational database when the key attributes of objects are available, consistent, and having error-free values. [16] presents an algorithm for generating EER schema from relational schema when key dependencies and key-based inclusion dependencies are known. Earlier approaches on schema translation such as [5], [10], [24] and a recent approach of [25] also belong to this class, since they only consider schema information and ignore data. Similar approaches have also been taken in the knowledge discovery and data mining (KDD) research community. For example, [14] applies an ID3-style algorithm to analyze frame-based data structures in the KATE system. [22] proposes a method for analyzing individual relations and combining the results across databases based on the known primary and foreign keys. Another approach, taken by [19] and [20], is to analyze the data fetching statements, such as SQL queries, embedded in application programs to infer objects and their relations. The limitation of this approach is that not all application programs are available for such an analysis.

Discovering and constructing conceptual models from relational databases without database schema information goes beyond the scope of reverse engineering. When the only information available is the given data stored in a database, the main issue becomes how to identify the objects, their keys and relationships before a reverse engineering process can begin. This is a typical knowledge discovery and datamining task, and little work has been done on the topic.

To the best of our knowledge, we are only aware of two previous works on discovering and constructing conceptual models from database instances with an emphasize on key identification. [21] lays out a general approach for model discovering from relational databases. However, the discussion in

[21] remained at a proposal level with little detail on how their system was actually developed. [17] presents a reverse engineering approach to analyze the dependencies in a database for the background domain knowledge. This approach first searches unique or near-unique attributes as keys, and then groups relevant attributes to form complex objects. However, it does not make use of the constraints that should apply to the model to be discovered to confine the key search process, so that the search space may be prohibitively large. The report does not show how well this approach will scale up to large size legacy databases. In a large legacy database, an exhaustive search of unique attributes may take too much computation resource and result in too many false key attributes.

3.3 Our approach and McKey system

In this paper, we develop a new approach for key identification and model construction in large legacy databases. Our approach is based on the observations that the process of key identification can be guided by the constraints posed by the underlying model. The main novelty of this approach is an iterative process to accommodate noisy data, and key identification under the constraints of the underlying model to reduce computation. By exploiting the model constraints, our approach can identify primary and foreign keys efficiently and build a conceptual model from a legacy data. Compared to the existing key identification and model construction techniques, our approach has many favorable features, including robustness in dealing with noise data, and scalability to large data sets.

To materialize our approach, we develop a system for Model Construction with Key identification called McKey. This system is able to build entity-relationship models from large instances of large legacy relational databases without database schema information. We apply McKey to three very large legacy databases. The models discovered by McKey are comprehensive with high accuracy. These conceptual models require several man-weeks of work when built with conventional database tools. In addition, the three models discovered and constructed by McKey are more complete than the ones that were previously developed manually.

3.4 Outline

The paper is organized as follows. Section 2 introduces the concepts of ER model and relational database as background information. Section 3 describes our approach and the McKey system in detail. Section 4 discusses the experimental results of applying McKey to three large legacy databases. Finally, Section 5 concludes the paper with a discussion of future work.

An extended abstract of this paper will appear in [23].

3.5 Background

In this section, we briefly describe entity-relationship model and relational database. We will only consider the concepts and definitions that will be used in the rest of the paper. Detailed information can be found in many database books, for example [15, 27].

3.6 Entity-relationship model

An *entity-relationship model*, or *ER model* for short, describes the conceptual schema or logical organization of the data under consideration. It views the data as consisting of entities and relationships between the entities. An *entity* is a thing that exists and is distinguishable from others. An entity is called a *strong entity* if it can exist independently of other entities, and is called a *weak entity* if its existence depends on another entity. A *relationship* connects entities. Entities and relationships combined are referred to as *objects*. The properties of an object are characterized by the *attributes* of the object. Each attribute has its own *domain*, which specifies the possible values the attribute may have. An object in an ER model has a unique name. While an object's attributes may not be unique and two objects may have attributes of identical names. Due to the distinguishability of objects, an ER model is generally considered as an object-oriented model.

An ER model is typically represented by a graph called *entity-relationship diagram*, or *ER diagram* for short. In an ER diagram, an entity is represented by a rectangle with entity's attributes linked with lines. A relationship in an ER diagram is represented by a diamond with lines linking its participating entities. One example of an ER diagram is shown in Figure 1 of Section 4.

3.7 Relational database

A *relational database* is defined as a tuple $(\mathcal{R}, \mathcal{F}, \mathcal{I})$, where $\mathcal{R}$ is the database schema, $\mathcal{F}$ a set of functional dependencies, and $\mathcal{I}$ a set of inclusion dependencies. Specifically, a *database scheme* $\mathcal{R}$ consists of a set of *relation schema* $R_i = (A_{i1}, A_{i2}, \cdots, A_{in})$, where A_{ij} are attributes of R_i. An instance of a relation schema is called a tuple with its values drawn from the domains of the corresponding attributes.

A *functional dependency* over a relation schema R is defined as an expression $R : X \rightarrow Y$, where $X \subseteq R$ and $Y \subseteq R$ are subsets of attributes in R. Let r be an instantiated relation over a relation schema R, $R : X \rightarrow Y$ holds in r if and only if for any pair of tuples t and t' in r, if $t[X] = t'[X]$, then $t[Y] = t'[Y]$. This means that under the functional dependency R, X can uniquely determine Y. Particularly, we refer X as a *superkey* of R if $R : X \rightarrow R$ holds. Furthermore, we call a superkey X as a *candidate key* of R if no subset of X is a superkey of R. By convention, we assume that one of the candidate keys of R is designated as the *primary key*, or *key* for short, of R. In other words, a key is a set of attributes whose values are sufficient for uniquely identifying the tuples of a table.

An *inclusion dependency* over $\mathcal{R}$ is defined as an expression $R_i[X] \subseteq R_j[Y]$, where R_i and R_j are relation schemas of $\mathcal{R}$, and X and Y are equal-length sequences of attributes of R_i and R_j, respectively. An inclusion dependency $R_i[X] \subseteq R_j[Y]$ holds if and only if every tuple in $R_i[X]$ is also in $R_j[Y]$. An inclusion dependency $R_i[X] \subseteq R_j[Y]$ is called *key based* if Y is a superkey of R_j. Moreover, if Y is a primary key of R_j, then the dependency is called *primary key based*, and X is called a *foreign key* of R_i. Conceptually, inclusion dependencies introduce a set of *reference integrity constraints* between R_i and R_j, because they specify how items in one relation should be referred to from another relation.

3.8 The Approach and McKey System

To reiterate, we are interested in discovering and constructing an entity-relationship (ER) model from an instance of a large relational database without any knowledge of the database schema. Specifically, what we are given is just the data stored in the format of tables. Without loss of generality, we assume that the given data can be accessed by SQL statements, which are supported almost by all well-known database menagement systems such as Oracle, Sybase and Informix databases.

3.9 The main ideas and overall process

The overall model discovery and construction process has three main steps. The first step is to discover objects that are implicitly embedded in a given data, including their identities, attributes, and associated relationships. This step, by its nature, is data driven and uses methods for data mining, as data-intensive computation is required to clean, group, and compare data items and to identify their features. The second step is to classify the objects into entity objects and relationship objects. In other words, this step is to classify the relations in a relational database into entity relations and relationship relations. Finally, the third step is to construct an ER model using the classified objects.

By referring to the descriptions in Section 2, the first two steps of the overall process combined are aimed to discover the underlying database schema $(\mathcal{R}, \mathcal{F}, \mathcal{I})$ of the given data. The third step is in fact a transformation from a database schema to an ER model. Furthermore, the last two steps combined are traditionally regarded as reverse engineering.

The novelty of our approach includes (1) using an iterative process to accommodate noisy data, and (2) applying the constraints of an underlying model, which is to be discovered, in the process of key identification to reduce computation.

3.10 Iterative process to accommodating noisy data

The data in a legacy database is typically noisy, i.e., some data item may be incorrectly recorded, modified, replaced, or even missing. Due to noise, a table in legacy database may have many duplicate tuples and it is also possible no subset of attributes of a table has unique values. This causes the difficulty of identifying primary keys from the data, as primary keys must have unique values in their corresponding tables. A similar problem occurs when searching for foreign keys. The first idea is to use a threshold in measuring the uniqueness of possible keys, in order to accommodate noisy data. This idea not only makes key identification feasible when data is noisy, but can also significantly speed up the process of key identification.

However, the value of the threshold used will affect the performance of the system, in terms of processing time and completeness of the model discovered. Although a large threshold may significantly reduce the computation, it may also fail to identify genuine keys or even fail to find keys, leading to incorrect or incomplete models. One solution to the problem of selecting a good threshold is to use domain knowledge. However, this defeats the assumption that domain experts are not available during the overall model building process. Furthermore, even if domain expertise is available, to find the right threshold that strikes the right balance between maximally reducing total computation and finding the most informative model is not an easy job.

In our McKey system, we choose an iterative approach to automatically search for a good threshold. Initially, a large threshold is used, and a database schema is discovered. If this schema is too simple, meaning that it contains too few primary and foreign keys and relationships, then the threshold is reduced and another round of schema discovery begins. This process continues until a satisfactory schema is discovered. During this process, a user may take the responsibility of judging the quality of a discovered schema.

We now discuss an iteration of model discovery and construction.

3.11 Speed up key identification with model constraints

For a relational database, the main task of the first step is to find the primary keys and foreign keys of a relation schema, as a table in a relational database can be directly converted to an object. This task, however, is not trivial because of the number of possible combinations of attributes that can constitute a key and the noise in the data. One straightforward method for finding a primary key from noisy data is to systematically search through all possible combinations of the attributes of a relation schema R to find such an attribute combination that has the highest percentage of unique values among all tuples of R. Assume that there are k relations, and each of them has m attributes and t tuples, then this process takes

$$k\left[\binom{m}{1}+\binom{m}{2}+\cdots\binom{m}{m}\right]t^2 = O\left(kt^2 2^m\right) \tag{1}$$

comparisons, as there are $2^m - 1$ nonempty attribute subsets and each requires t^2 comparisons to determine the number of unique tuples. To find all inclusion dependencies in a database, a similar and straightforward method is to search every pair of relation schemas R_i and R_j, and check all possible pairs of attribute subsets between the two relations to see if one's values are contained in the other. Assume that there are k relations, each with m attributes and t tuples, then this process will take

$$\binom{k}{2}((2^m - 1)t)^2 = O\left(k^2 t^2 2^m\right) \qquad (2)$$

comparisons. (1) and (2) indicate that the complexity of these straightforward methods grows exponentially with the number of attributes of a relation schema, making them infeasible in practice on large databases in which the number of attributes in a relation schema is typically large.

The problem of the above simple methods is that they neglect the constraints imposed by the underlying conceptual data model. These constraints can play important roles in reducing the complexity of searching for keys. For example, if information of candidate keys of relations are known, it can be used to restrict the search of foreign keys to those that point to candidate keys. Similarly, if possible foreign keys are known, they can be used to speed up the process of searching for primary keys by eliminating those candidate keys that are not pointed by a possible foreign key.

Therefore, the second idea to reduce the total computation in searching primary and foreign keys, and therefore the overall process of model discovery and construction, is to use the constraints imposed by the underlying conceptual model which is to be found.

3.12 Overview of individual steps

In the McKey system, one iteration of model construction consists of three major steps with the first one having four substeps. These steps, including their constraints and actions, are listed in Table 1.

As shown in Table 1, Step 1.1 identifies all possible candidate keys whose uniqueness exceeds a threshold under the constraint that keys must have unique values. Step 1.2 is to discover possible foreign keys under the constraint that foreign keys must point to keys. This constraint restricts the search to only those candidates that point to possible keys identified in Step

Step	*Constraints and conditions*	*Actions*
1.1	Keys are likely unique	Find possible keys
1.2	Foreign keys reference to keys	Find possible foreign keys
1.3	Keys referenced by foreign keys	Prune spurious keys
1.4	Foreign keys reference to keys	Prune spurious foreign keys
2	Key reference structure	Find entities and relationships
3	Identified database schema	Construct a complete ER model

Table 1: Steps for model construction with key identification.

1.1, which significantly reduces computation. Step 1.3 refines the set of possible keys by eliminating those that are not referenced by any possible foreign key. This is under the constraint that if a key is primary, then it is very likely referenced by a foreign key. In other words, given two possible keys that one is pointed by a foreign key and the other is not, we will prefer the former to be the primary key. Step 1.4 refines the set of possible foreign keys by eliminating those that are not pointing to any possible primary key identified in the previous steps. This is based on the constraint that foreign keys must serve as references between entities in the final model. In Step 2, the identified possible primary keys and foreign keys are used as constituents to classify the database relations into appropriate objects of entities and relationships. Step 3 carries out the final construction of an ER model based on these classifications.

In the rest of this section, we describe each step in detail.

3.13 Step 1: Key identification

The goal of this step is to discover the primary and foreign keys of the tables in a database without the database schema.

3.14 Step 1.1: Identifying possible primary keys

By definition, candidate keys must have unique values because they can unambiguously identify objects in a database. Thus, identifying candidate keys implies to evaluate the "uniqueness" of attribute subsets. Let X be a set of attributes of a relation schema R, $|R|$ the total number of tuples in R, $v(X)$ the number of tuples of distinguishable values of X. We can then define the uniqueness of X as

$$\mu(X, R) = \frac{v(X)}{|R|} \tag{3}$$

Due to noise, there is no guarantee that every table in a database has a key X with its uniqueness $\mu(X, R) = 1.0$. If the table was indeed developed from a relation schema R, there must exist a threshold η, for $0 < \eta < 1$, such that if X is a primary key of R, then $\mu(X, R) > \eta$.

Given a relation R and a uniqueness threshold η, there are two methods that we can use to find an attribute set X with $\mu(X, R) > \eta$. Both methods have their strength and weakness. The strength of the first method is its simplicity. It simply counts the unique tuples in every possible subset of attributes $X \subseteq R$. For a subset of attributes $X = (X_1, X_2, \cdots, X_k)$ of a relation R, one SQL query is sufficient to find the number of the unique tuples in X, as follows:

```
SELECT COUNT(*)
FROM (SELECT DISTINCT R.X_1, R.X_2, ..., R.X_k FROM R)
```

In essence, this method pushes all the implementation details to the SQL processor and the resulting program is very clean and understandable. The drawback of this method, however, is that it is not efficient, especially for large tables, since the SQL query has to be issued against all possible subset of attributes $X \subseteq R$. Specifically, the computation of the uniqueness of each possible subset must start from scratch, resulting in an $O(2^m)$ computation, where m is the number of attributes of R.

The second method is introduced to reduce the computation cost of the first method by saving the values of the uniqueness of small sets of attributes for later use. The strength of this method is its efficiency for large tables, while its weakness is the associated complexity.

This method is based on the observation that the uniqueness of a set of attributes can be computed by using the number of duplicate values of the attributes. The method is built upon the concept called *duplicate itemset*, which is a special case of itemset of [1]. For a given table, denote the duplicate itemset of an attribute set X as *Dup*(X), and a duplicate item of X as

$< x, dup(x) > \in Dup(X)$, where x is a value of X and $dup(x) > 1$ is the number of tuples in the table whose attributes in X have value x. The condition $dup(x) > 1$, which means that there must be more than one tuples that have the same value, specifies a duplicate. If X has a total of k duplicate items, x_i for $i = 1, 2, \cdots, k$, then the number of tuples of distinguishable values of X in the table, which is $v(X)$, can be computed as

$$v(X) = |R| - dup(x_1) - dup(x_2) - \cdots - dup(x_k) + k \tag{4}$$

where $|R|$ is the total number of tuples in relation R. To make it concrete, consider $X = A$ in a relation $R = (A, B, C)$ as shown in Table 2. A has two duplicate items, $< [1], dup[1] = 2 >$ and $< [2], dup[2] = 2 >$. Following (4), we then have $v(A) = 4 - dup[1] - dup[2] + 2 = 2$, where 4 is the number of tuples in the table.

A	B	C
1	4	3
2	5	3
1	3	2
2	5	5

Table 2: An example table for computing uniqueness.

The total computation can be reduced in this method because the duplicate itemset of an attribute set Y can be built on the duplicate itemset of $X \subset Y$. This is based on the fact that if two tuples can be aggregated into a same duplicate item of Y, they must have the same values on Y, leading to the fact that these tuples must also have the same values on X, which is a subset of Y. One by one, the method extends a duplicate item of X to duplicate items of Y. Consider Table 2 again as an example. Given $D(A) = \{< [1], 2 >, < [2], 2 >\}$, we now consider $D(A, B)$. Item $< [1], 2 > \in D(A)$ may be extended to items $< [1, 3], dup([1, 3]) >$, $< [1, 4], dup([1, 4]) >$ and $< [1, 5], dup([1, 5]) >$, since 3, 4 and 5 are possible values of B. However, $dup([1, 3]) = dup([1, 4]) = 1$ and $dup([1, 5]) = 0$, thus item $< [1], 2 > \in D(A)$ cannot be extended to the ones in $D(A, B)$ since $dup(x) > 1$ must hold. Similarly, item $< [2], 2 > \in D(A)$ may be extended to items $< [2, 3], dup([2, 3]) >$,

$< [2,4], dup([2,4]) >$ and $< [2,5], dup([2,5]) >$. Item $< [2,5], 2 >$ is the only valid item of $D(A, B)$. Thus, $v(A, B) = 4 - dup([2,5]) + 1 = 3$.

In summary, the second method computes the uniqueness of attribute subsets, incrementally from a small subset to a large one by sequentially extending the duplicate itemset of the small subset to that of the large subset. The resulting algorithm is similar to the Apriori algorithm of [1].

Using one of the above methods of computing uniqueness, the search for possible primary keys for a given relation R proceeds through all attribute subsets of R in an increasing order of their size. The ones whose uniqueness exceed the threshold η are reserved. In practice, the maximum number of attributes in an attribute subset should be restricted to some number, which is 5 in the McKey system. This is because according to the design principles of relational database, the possibility of having a compound key with more than 5 attributes is very low. In addition, since legacy databases may contain much noisy data, some relations may have no attribute subset that satisfies the threshold η. Such a relation should be reported to human analyzer for further investigation.

As one may expect, the result of this step is only a set of possible candidate keys for every relation. They must be refined or filtered further before primary keys can be identified.

3.15 Step 1.2: Identifying possible foreign keys

Given the candidate keys found in the previous step, the McKey system then searches for possible foreign keys based on the constraint that foreign keys must refer to primary keys. In particular, given two relations R_x and R_y, we will only search pairs of attribute sets (X, Y) such that Y is a possible primary key for R_y, and X has the same types of attributes as the attributes in Y. This constraint can greatly reduce the computation for foreign keys.

Note that in searching a pair of attribute sets (X, Y) for possible foreign keys, an attribute in X does not have to have the exactly the same name as its counterpart in Y. The only requirement for pairing two attributes is that their domains are the same. This can be simply done by comparing the data types of the corresponding columns of two involved database tables.

Legacy databases may not have foreign keys that completely satisfy the definition of foreign keys in theory due to noise in data. To overcome the difficulty caused by noisy data, we then introduce a *fitness* to measure the likehood an attribute set being a foreign key. Let X be a subset of attributes of R_x, Y a subset of attributes of R_y, and X and Y contain the same type of attributes, we then define the fitness of X being a foreign key referring to Y as follows,

$$f(X, Y) = \frac{|X \cap Y|}{|X|} \tag{5}$$

where $|X|$ is the number of distinct tuples in X, and $|X \cap Y|$ is the size of the intersection between X and Y. To implement this function, let $X = (X_1, X_2, \cdots, X_k)$ be a subset of attributes of relation R_x, $Y = (Y_1, Y_2, \cdots, Y_k)$ a subset of attributes of relation R_y, we use the following SQL statement to obtain the value of $|X \cap Y|$,

```
SELECT COUNT(*)
FROM (SELECT DISTINCT $R_x.X_1, R_x.X_2, \cdots, R_x.X_k$
      FROM $R_x$ WHERE $R_x.X_1, R_x.X_2, \cdots, R_x.X_k$
              IN (SELECT $R_y.Y_1, R_y.Y_2, \cdots, R_y.Y_k$ FROM $R_y$))
```

The value of $f(X, Y)$ can be subsequently computed by (5).

Similar to the reason of using threshold η on the uniqueness of a primary key to tolerate noisy data, we introduce a threshold ζ for the fitness of a foreign key. That is, an attribute set will be reserved as a possible foreign key if its fitness is greater than the threshold ζ, i.e., $f(X, Y) > \zeta$.

3.16 Step 1.3: Eliminating spurious primary keys

Up to this point, the knowledge accumulated for model construction includes the possible primary keys and possible foreign keys. Interestingly enough, the information about primary and foreign keys can be used to eliminate those possible primary keys that should not be considered as candidates for genuine primary keys. To this end, a "spurious" key is defined as an identified possible primary key that is not pointed to by any one of the possible foreign keys. The reason behind this key elimination is that if both X and Y are identified possible primary keys for a relation R, and X is referenced by a foreign key while Y is not, then Y can be eliminated from the set of possible primary keys for R. The justification of this key elimination is that if a candidate key is genuine, then it is referenced by a relation when such a reference exists. When two or more possible primary keys are referenced by foreign keys, the one that is referenced most frequently will be selected as the primary key for the relation.

3.17 Step 1.4: Eliminating spurious foreign keys

By definition, a foreign key of a relation must reference to the primary key of another relation. Not every possible foreign key found in Step 1.2 satisfies this condition. Therefore, a possible foreign key that does not reference to a primary key can be removed since they will not be considered in the final model.

3.18 Step 2: Classifying relations

In this step, each relation in the given relational database will be classified as either an entity relation or a relationship relation, which is to be converted into an entity or a relationship in the ER model, respectively. The classification of a given relation will be based on the collective information of the relation's primary keys and foreign keys. Whether a relation R should be classified as an entity or a relationship is based on the interdependency between its primary key and foreign keys. The heuristics used in this section are adopted from the ones proposed in [9], with some minor modifications. Specifically, the decision is made based on the following heuristics:

Heuristic 1 *If the primary key of a relation R does not contain any foreign key, then R is classified as an entity relation.*

Heuristic 1 is used to identify the entity relations that are independent of other objects, since they do not have a foreign key pointing to other objects.

Heuristic 2 *If a relation R_i has only one foreign key which is a subset of the primary key of another relation R_j, then R_i is classified as an entity relation.*

The rationale behind Heuristic 2 is that if the primary key of a relation R_i is a subset of the primary key of another relation R_j, then it is very likely that R_i is a sub-object of R_j, therefore, should be considered as an entity.

Heuristic 3 *If a relation R has two foreign keys and their union is the primary key of R, then R can be classified as a relationship relation. This relationship links the two entity relations pointed to by the foreign keys.*

Heuristic 3 means that, since a relation's primary key is the union of two foreign keys, the existence of this relation is very likely to link those two data items referenced by the foreign keys. Thus, such a relation should be classified as a relationship.

Heuristic 4 *If a relation R has two foreign keys and one is in the primary key and the other is not, then classify R as an entity relation. This entity will have two relationships through the foreign keys.*

The rationale for Heuristic 4 is similar to that of Heuristic 2, because if a part of a relation R_i's primary key is enough to identify another relation R_j, then R_i is very likely to be a sub-object of R_j.

Heuristic 5 *If a relation R has n foreign keys, for $n > 2$, and at least one of them is contained in the primary key of R, then R is classified as an n-ary relationship relation, with links to the n entity relations pointed to by the foreign keys.*

To illustrate the rationale for Heuristic 5, consider the following example,

```
MAN(SSN, Name, Bdate);
WOMAN(SSN, Name, Bdate);
LAWYER(SSN, Name, Bdate);
DIVORCE(ManSSN, WomanSSN, LawyerSSN, Date).
```

The primary key of DIVORCE is ManSSN (or WomanSSN) and it has three foreign keys referring to MAN, WOMAN, and LAWYER, respectively. Then the relation DIVORCE should be classified as a 3-ary relationship among MAN, WOMAN, and LAWYER entities.

3.19 Step 3: Constructing an ER model

Up to this point, a database schema $(\mathcal{R}, \mathcal{F}, \mathcal{I})$ has been discovered. This step is to transform the schema into an ER model. Note that the translation is at the schema level without data analysis involved.

Specifically, All entity relations will be converted into entities, and their names and attributes (except the foreign keys) will be carried over as the attributes of the corresponding entities. All the relationship relations will be converted into relationship, their foreign keys will be used as pointers to the appropriate entities, and their attributes will be made as attributes of the relationship. Whenever possible, entities and relationships and their attributes will use the same name as they were in the relational database.

3.20 Experiments and Applications

To evaluate our approach, we have applied the McKey system to build ER models of the instances of three real large legacy databases: ALPI, CVI and GITI8I. These database instances include data collected from some logistics applications. We need conceptual models of these database instances to build a large model of the domain of an application in which information from multiple data sources, including ALPI, CVI and GITI8I, needs to be integrated to provide a unified, overall view of the application [4]. All of these databases have the typical characteristics of legacy database: Their underlying schemas are complex, and they contain large amounts of data. Specifically, ALPI has 67 tables, with maximum 60 attributes and 515,668 rows; CVI has 104 tables, with maximum 38 attributes and 99,264 rows; and GITI8I has 48 tables, with maximum 20 attributes and 1,480,445 rows. Furthermore, the database schemas, which are classified information, are not available and domain experts are almost impossible to access. In addition, the data in these database instances are noisy. For example they contain duplicates and have missing items. It is also very possible that some of the data items in these three database instances were misrecorded.

We have previously built conceptual models of these database instances manually. The overall manual modeling took about a few weeks of an experienced knowledge engineer's work through painstaking probing and guessing on the data stored in the database instances. These manually built models

Table name	*Rows*	*Columns*	*Possible primary key*	*Uniqueness*
Database_History	36	10	Tablename	0.916667
			Tablename, Username	0.916667
			Tablename, Timestamp	1.000000
			Tablename, Mod_Type	0.916667
			Tablename, Datafile	1.000000
Logad_Tables	120	9	Table_Name	0.983471
			Pretty_Name	1.000000
Nsn_Description	25716	60	Niin	1.000000
			Nsn	1.000000

Table 3: Examples of possible primary keys identified in ALPI database.

are not complete. We also do not have a high confidence on the correctness of these models because of the amount of data involved and the noise in the data.

The purpose of applying McKey to these database instances is to discover more complete and comprehensive models for the application. In the future, we plan to use McKey to discover and construct conceptual models of new data sources, but only rely on knowledge engineer to verify and validate the discovered models.

We now discuss the results on ALPI in the rest of this section.

Table 3 lists the possible primary keys of three selected database tables of ALPI found by one iteration of McKey. The uniqueness threshold η for primary key identification is set to 0.8. As illustrated by Table 3, the tables in this database have many possible primary keys. Although from uniqueness of data items in a table, many possible primary keys could be selected as the primary key of the table, the genuine primary key needs to be distinguished from the other candidates based on foreign key reference structure, which is partially shown in Table 4. Specifically, for each possible primary key in Table 3, Table 4 lists the foreign keys that reference this possible primary key. As observed from the experimental data which is partially shown in Tables 3 and 4, the foreign key referencing structure in this database is well organized and reserved. Most foreign keys point to the same possible primary key if they refer to the same relation. Using the information of foreign key

Table name	*Possible primary key*	*Referenced by*
Database_History	Tablename	Logad_Table_Segments(Table_Name)
	Tablename, Username	
	Tablename, Timestamp	
	Tablename, Mod_Type	
	Tablename, Datafile	
Logad_Tables	Table_Name	Database_History(Tablename)
		Glad_Meta_Join_From(Table_Name)
		Glad_Meta_Pretty_Names(Table_Name)
		Logad_Codes(Table_Name)
		Logad_Columns(Table_Name)
		Logad_Indices(Table_Name)
		Logad_Post_Load_Proc(Table_Name)
		Logad_Table_Segments(Table_Name)
	Pretty_Name	
Nsn_Description	Niin	
	Nsn	Cif_Failure_Factor(Ei_Nsn)
		Tav_Stock_Report_1(Nsn)
		Tav_Stock_Report_2(Nsn)

Table 4: Using foreign key reference structure to identify genuine primary keys.

references, genuine primary keys can be easily identified. For example, both Pretty_Name and Table_Name have been recognized as possible primary keys for table Logad_Tables. However, Table_Name has a total of eight foreign key references, while Pretty_Name has no foreign key reference at all. Therefore, Table_Name is chosen as the primary key of table Logad_Tables. Note that if uniqueness were the sole criterion for selecting genuine primary keys, then Table_Name would be less qualified for the primary key than Pretty_Name, since the uniqueness of Table_Name, which is 0.983477, is less than that of Pretty_Name, which is 1.0. This simple example shows that the constraints from an underlying model indeed played a crucial role for key identification in the McKey system. Similar situations occur for table Database_History and table Nsn_Description. Furthermore, the system is able to identify a possible foreign key even if its name is not the same as the possible primary key

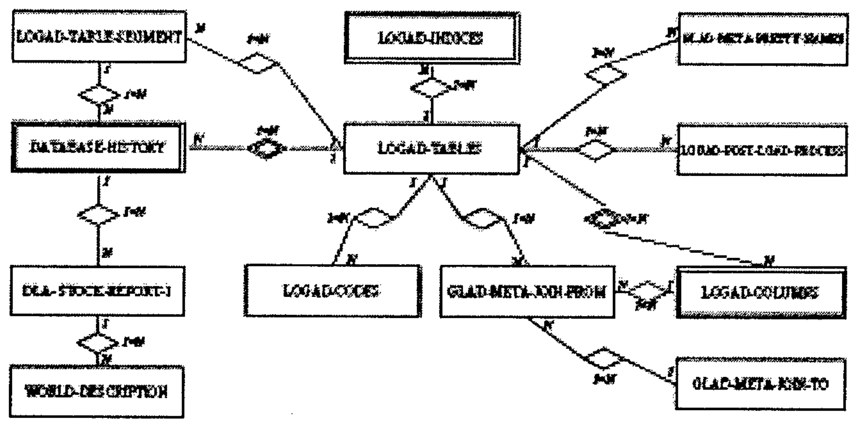


Figure 1: A fragment of a discovered ER model of ALPI database.

which it is referencing to, since in searching for possible foreign keys, the system only considers the data types of a pair of attribute sets. One such example is shown in Table 4, in which the possible foreign key Tablename in table Database_History has a different name from the possible primary key Table_Name in table Logad_Tables.

For the ALPI database, McKey has discovered and constructed an ER model of 67 entities and 25 relationships. A small part of this model is shown in Figure 1. The entire process takes about 3 hours on an HP 710 machine. Compared to its manually built counterpart, this machine-discovered model is not only created quickly, saving several day's work of a knowledge engineer, but also much more complete and comprehensive. Similar observation holds on CVI and GITI8I databases.

Conclusions and Future Work

Discovering a conceptual object model from large relational databases is an important yet difficult task with many applications in practice. In this paper, we presented an approach for discovering and constructing an ER model from instances of large relational databases with no knowledge of underlying database schemas. Compared to the existing methods, this approach is data-driven and constraint-based. The important features of our approach include robustness in dealing with noisy data and scalability to large databases and large data sets. We implemented our approach in a system called McKey, which stands for Model Construction with Key identification. We tested McKey on instances of three real, very large legacy databases. The results obtained are very promising. For the three large legacy databases, the McKey system is able to discover and construct more complete and more comprehensive models than the ones created manually, which gives many magnitudes of savings of manpower.

The future work will focus on further improving McKey's performance and extending its functionality and applicability. Specifically, we are currently working on coupling the McKey system with an embedded relational database to minimize the amount of data access and increase computation efficiency. We are also extending McKey to accommodate the Open Database Connectivity (ODBC) to expand McKey's applicability to any relational database supporting ODBC. In addition, we are also developing a graphical interface of McKey to support the functionality of graphically viewing and browsing through the discovered conceptual models as well as support direct human intervention during the discovery process.

We expect the final, enhanced McKey system to be a handy tool for knowledge discovery and datamining from databases, and plan to release the software to the research community in the near future.

Acknowledgements

Thanks to Jafar Adibi for drawing the piece of ER model shown in Figure 1, and to Marissa Contreras for proofreading this paper.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20-th Conference on Very Large Databases (VLDB-94)*, pages 487–499, Santiago de Chile, Chile, September 12-15 1994.

[2] Y. Arens, C. Y. Chee, C-N. Hsu, and C. A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal of Intelligent and Cooperative Information Systems*, 2:127–158, 1993.

[3] Y. Arens, C. A. Knoblock, and W-M. Shen. Query reformulation for dynamic information integration. *Journal of Intelligent Information Systems*, 6:99–130, 1996.

[4] Y. Arens, W. Zhang, Y. Lee, J. Dukes-Schlossberg, and M. Zev. Warfighter's information packager. In *Proceedings of the 10-th Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, pages 1095–1100, Madison, WI, July 26-30 1998.

[5] D. Boulander and S. T. March. An approach to analyzing the information content of existing databases. *Database*, pages 1–8, 1989.

[6] R. J. Brachman. "I lied about the trees" or, defaults and definitions in knowledge representation. *AI Magazine*, 6:80–93, 1985.

[7] P. P. Chen. The entity relationship model - toward a unified view of data. *ACM Transactions on Database Systems*, 1:9–36, 1976.

[8] R. H. L. Chiang. A knowledge-based system for performing reverse engineering of relational databases. *Decision Support Systems*, 13:295–312, 1995.

[9] R. H. L. Chiang, T. M. Barron, and V. C. Storey. Reverse engineering of relational databses: Extraction of an EER model from a relational database. *Data and Knowledge Engineering*, 12:107–142, 1994.

[10] K. H. Davis and A. K. Arora. Converting a relational database model into an entity-relationship model. In *Proceedings of the 6-th International Conference on Entity-Relationship Approach*, pages 271–283, New York, NY, November 9-11 1987.

[11] M. Ginsberg. *Essentials of Artificial Intelligence.* Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[12] T. Landers and R. L. Rosenberg. An overview of Multibase. In H. J. Schneider, editor, *Distributed Data Bases.* North-Holland, 1982.

[13] A. Y. Levy, D. Srivastava, and T. Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems, Special Issue on Networked Information Discovery and Retrieval*, 5, 1995.

[14] M. Manago and Y. Kodratoff. Construction of decision trees from complex structured data. In Piatetsky-Shapiro and Frawley, editors, *Knowledge Discovery in Databases*, pages 289–308. AAAI Press, 1991.

[15] H. Mannila and K-J. Räihä. *The Design of Relational Databases.* Addison-Wesley, 1992.

[16] V. M. Markowitz and J. A. Makowsky. Identifying extended entity-relationship object structures in relational schemas. *IEEE Transaction on Software Engineering*, 16:777–790, 1990.

[17] S. Mckearney and H. Roberts. Reverse engineering databases for knowledge discovery. In *Proceedings of the* 2^{nd} *International Conference on Knowledge Discovery & Data Mining (KDD-96)*, pages 375–378, Portland, Oregon, August, 2-4 1996.

[18] M. Minsky. A framework for representing knowledge. In P. Winston, editor, *The Psychology of Computer Vision*, pages 211–277. McGraw Hill, New York, 1975.

[19] J-M. Petit, J. Kauloumdjian, J-F. Boulicaut, and F. Toumani. Using queries to improve database reverse engineering. In *Proceedings of 13-th International Conference on Entity-Relationship Approach, Lecture Notes in Computer Science*, volume 881, pages 369–386. Manchester, UK, December 1994.

[20] J-M. Petit, F. Toumani, J-F. Boulicaut, and J. Kauloumdjian. Towards the reverse engineering of denormalized relational databases. In *Proceedings of 12-th International Conference on Data Enginnering*, New Orleans, Louisiana, Febuary 1996.

[21] W. J. Premerlani and M. R. Blaha. An approach for reverse engineering of relational databases. *Communications of the ACM*, 37:42–49, May 1994.

[22] J. S. Ribeiro, K. A. Kaufmann, and L. Kerschberg. Knowledge discovery from multiple databases. In *Proceedings of the* 1^{st} *International Conference on Knowledge Discovery & Data Mining (KDD-95)*, pages 240–245, Montréal, Québec, Canada, August 20-21 1995.

[23] W-M. Shen, W. Zhang, X. Wang, and Y. Arens. Model construction with key identification. In *Proceedings of SPIE Conference On Data Mining and Knowledge Discover - Theory, Tools, And Technology*, Orlando, Florida, 5-9 April 1999, to appear.

[24] F. Springsteel and C. Kou. Reverse data engineering technology for visual databsae design. *Information and Technology*, 1992.

[25] Z. Tari, O. Bukhres, J. Stokes, and S. Hammoudi. The reengineering of relational databases based on key and data correlations. In S. Spaccapietra and F. Maryanski, editors, *Searching for Semantics: Data Mining, Reverse Engineering, etc.* Chapman & Hall, 1998.

[26] Z. Tari, K. Yetongnon W. Cheng, and I. Savnik. Towards cooperative databases: The DOK approach. In *Proceedings of International Conference on Parallel and Distributed Computing Systems (PDCS-96)*, pages 595–600, Dijon, 1996.

[27] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume I. Computer Science Press, Rockville, MD, 1988.

4 SIMS PORT TO C++

A port of SIMS to C++ was one of the foundations of our overall effort to increase the usability of our system. The process was carried out in a series of phases, each of which provided more functionality. The process of porting SIMS to C++ was planned to satisfy the following design goals:

- The resulting C++ system should be efficient, robust and extensible. A phased approach had to be followed to allow for incremental extension of the system's functionality. Each phase had to result in a complete working system.
- Interoperability with the existing Lisp system had to be maintained so that the modules of the original Lisp system could be used in place of the C++ system modules, and vice versa. Maintaining interoperability facilitated smooth transferring of new technologies to the C++ system, and testing the port at various phases to ensure its success.
- The system was designed to facilitate an eventual port to Java. We chose an object-oriented design that would make it relatively easy to transition to Java. We have no current plans for a Java port, but we foresee this as a possible path for future development, due to the multi-platform nature of Java and the strong possibility of its eventual inclusion in DII/COE.

The phased approach was implemented as follows:

	Multi-source access capabilities	**Query language**	**Plan optimization**	**Failure recovery**
Phase 1	yes	sources must be explicitly specified	no	limited, substitute replicated sources
Phase 2	yes	sources must be explicitly specified	yes	full SIMS recovery capabilities
Phase 3	yes	full SIMS language	yes	full SIMS recovery capabilities

4.1 CSIMS System Manual

A complete CSIMS system manual is attached to this report.

5 REFERENCES

Ambite, J.L. and Knoblock, C.A. 1997. Planning by rewriting: Efficiently generating high-quality plans. In Proceedings of AAAI-97.

Ambite, J.L. and Knoblock, C.A. 1998. Flexible and scalable query planning in distributed and heterogeneous environments. In Proceedings of AIPS-98.

Ambite, J.L.; Knoblock, C.A.; Muslea, I.; and Philpot, A. 1998. Compiling source descriptions for efficient and extensible information integration. Technical report, USC Information Sciences Inst.

Arens, Yigal, Craig A. Knoblock, and Wei-Min Shen. 1996. Query Reformulation for Dynamic Information Integration. Journal of Intelligent Information Systems, Vol. 6, 1996, pp. 99-130.

Arens, Yigal, Chin Y. Chee, Chun-Nan Hsu, Hoh In and Craig A. Knoblock. 1994. Query Processing in an Information Mediator. In Proceedings of the ARPA/RL Knowledge-Based Planning and Scheduling Initiative Workshop. Tucson, AZ, February 21–24, 1994.

Arens, Yigal, Chin Y. Chee, Chun-Nan Hsu and Craig A. Knoblock. 1993. Retrieving and Integrating Data from Multiple Information Sources. International Journal of Intelligent and Cooperative Information Systems. Vol. 2, No. 2, 1993. pp. 127–158.

Bayardo Jr., R.J.; Bohrer, W.; Brice, R.; Cichocki, A.; Fowler, J.; Helal, A.; Kashyap, V.; Ksiezyk, T.; Martin, G.; Nodine, M.; Rashid, M.; Rusinkiewicz, M.; Shea, R.; Unnikrishnan, C.; Unruh, A.; and Woelk, D. 1997. InfoSleuth: Agent-based semantic integration of information in open and dynamic environments. In Proceedings of ACM SIGMOD-97.

Blum, A. and Furst, M. 1995. Fast planning through planning graph analysis. In Proceedings of IJCAI-95.

Cohen, W.W. 1998. Integration of Heterogeneous Databases without Common Domains using Queries Based on Textual Similarity. In Proceedings of ACM SIGMOD-98.

Doorenbos, R.B.; Etzioni, O.; and Weld, D.S. 1997. A scalable comparison-shopping agent for the worldwide web. In Proceedings of the First International Conference on Autonomous Agents.

Genesereth, M.R.; Keller, A.M.; and Duschka, O.M. 1997. Infomaster: An information integration system. In Proceedings of ACM SIGMOD-97.

Hammer, J.; Garcia-Molina, H.; Nestorov, S.; Yerneni, R.; Breunig, M.; and Vassalos, V. 1997. Template-based wrappers in the TSIMMIS system. In Proceedings of ACM SIGMOD-97.

Huang, T. and Russell, S. 1997. Object identification in a Bayesian context. In Proceedings of IJCAI-97.

Kautz, H. and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In Proceedings of AAAI-96.

Knoblock, C.A. 1995. Planning, executing, sensing, and replanning for information gathering. In Proceedings of IJCAI-95.

Kushmerick, N. 1997. Wrapper Induction for Information Extraction. PhD thesis, Computer Science Dept., University of Washington.

Kwok, C.T. and Weld, D.S. 1996. Planning to gather information. In Proceedings of AAAI-96.

Levy, A.Y.; Rajaraman, A.; and Ordille, J.J. 1996. Query-answering algorithms for information agents. In Proceedings of AAAI-96.

MacGregor, R. 1988. A deductive pattern matcher. In Proceedings of AAAI-88.

Muslea, I.; Minton, S.; and Knoblock, C.A. 1998. Wrapper induction for semistructured, web-based information sources. In Proceedings of the CONALD-98 Workshop on Learning from Text and the Web.

Wiederhold, G. 1996. Intelligent Integration of Information. Kluwer.

The CSIMS Manual, v. 1.0

The CSIMS Manual Version 1.0*

José-Luis Ambite
Yigal Arens
Naveen Ashish
Craig A. Knoblock
Steven Minton
Jay Modi
Maria Muslea
Andrew Philpot
H. Jean Oh
Wei-Min Shen
Sheila Tejada
Weixiong Zhang
Information Sciences Institute and Department of Computer Science
University of Southern California
4676 Admiralty Way,
Marina del Rey, CA 90292, U.S.A.

May 27, 1999

Abstract

SIMS provides intelligent access to heterogeneous, distributed information sources, while insulating human users and application programs from the need to be aware of the location of the sources, their query languages, organization, size, etc.

This manual explains how to bring up a SIMS information server in a new application domain. After providing a short overview of relevant features of the SIMS system, it describes the modeling and programming work that has to be performed to support the extension of SIMS to a given collection of information sources in the domain. To aid a user inexperienced with the technological infrastructure underlying SIMS, the manual contains examples structured as a tutorial that can be followed to actually produce a working SIMS system.

*The research reported here was supported in part by Rome Laboratory of the Air Force Systems Command and the Defense Advanced Research Projects Agency under Contracts Number F30602-94-C-0210, F30602-97-2-0352, and F30602-97-2-0238 and in part by a grant from Computing Devices International. The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing the official opinion or policy of RL, DARPA, the U.S. Government, or any person or agency connected with them.

1 Introduction

The overall goal of the SIMS project is to provide integrated access to information distributed over multiple, heterogeneous sources: databases, knowledge bases, flat files, Web pages, programs, etc. In providing such access, SIMS tries to insulate human users and application programs from the need to be aware of the location of sources and distribution of queried data over them, individual source query languages, their organization, data model, size, and so forth. The processing of user requests should be robust, capable of recovery from execution-time failures and able to handle and/or report inconsistency and incompleteness of data sources. At the same time SIMS has the goal of making the process of incorporating new sources as simple and automatic as possible.

The SIMS approach to this integration problem has been based largely on research in Artificial Intelligence; primarily in the areas of knowledge representation, planning, and machine learning. A model of the application domain is created, using a knowledge representation system to establish a fixed vocabulary for describing objects in the domain, their attributes and relationships among them. Using this vocabulary, a description is created for each information source. Each description indicates the data-model used by the source, the query language, network location, size estimates, etc., and describes the contents of its fields in relation to the domain model. SIMS' descriptions of different information sources are independent of each other, greatly easing the process of extending the system. Some of the modeling is aided by source analysis software developed as part of the SIMS effort.

Queries to SIMS are written in a high-level language (Loom or a subset of SQL) using the terminology of the domain model — independent of the specifics of the information sources. Queries need not contain information indicating which sources are relevant to their execution or where they are located. Queries do not need to state how information present in different sources should be joined or otherwise combined or manipulated.

SIMS uses a planner to determine how to identify and combine the data necessary to process a query. In a pre-processing stage, all data sources possibly relevant to the query are identified. The planner then selects a set of sources that contain the queried information and generates an initial plan for the query. This plan is repeatedly refined and optimized until it meets given performance criteria. The plan itself includes, naturally, sub-queries to appropriate information sources, specification of locations for processing intermediate data, and parallel branches when appropriate. The SIMS system then executes the plan. The plan's execution is monitored and replanning is initiated if its performance meets with difficulties such as unexpectedly unavailable sources. It is also possible for the plan to include explicit replanning steps, after reaching a state where more is known about the circumstances of plan execution.

Changes to information sources are handled by changing source descriptions only. The changes will automatically be considered by the SIMS planner in producing future plans that utilize information from the modified sources. This greatly facilitates extensibility.

The rest of this section presents an overview of SIMS and its architecture. In Section 2 we show the format of the queries that a user would input to SIMS and the output that should be expected. Then we consider in more detail the specification of the domain model, in Section 3, and how information sources are described to the system, in Section 4. Section 8 gives a brief introduction on how to construct a wrapper for a new information source and how to communicate with the wrapper. Section 10 explains how to run SIMS both through its graphical user interface and its functional interface. Section ?? describes how to test and debug a new SIMS application. Section 11 presents the installation and system requirements. Finally, in Section 12 we show the code that would implement the example that is discussed throughout the manual. Section 13 contains a reading list of relevant papers.

1.1 Architecture and Background

A visual representation of the components of CSIMS is provided in Figure 1.

CSIMS addresses the problems that arise when one tries to provide a user familiar only with the general domain with access to a system composed of numerous separate data- and knowledge-bases.

Specifically, CSIMS does the following:

- **Modeling:** It provides a consistent way of describing information sources to the system, so that data

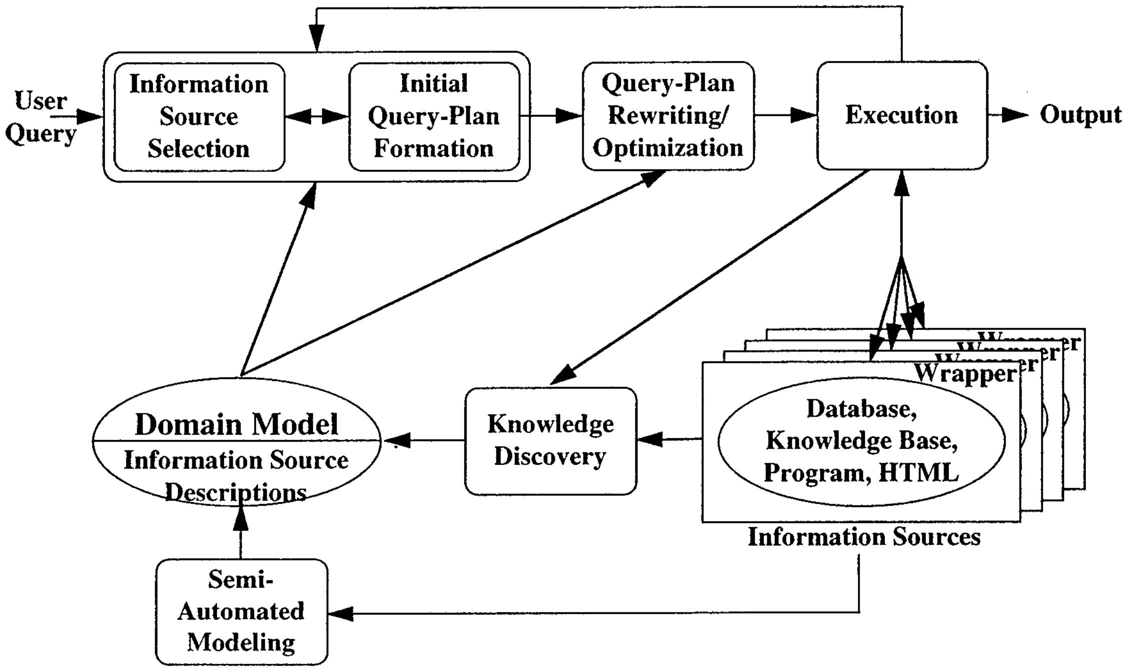

User Query
Information Source Selection
Initial Query-Plan Formation
Query-Plan Rewriting/ Optimization
Execution
Output
Domain Model
Information Source Descriptions
Knowledge Discovery
Wrapper
Database, Knowledge Base, Program, HTML
Information Sources
Semi-Automated Modeling

Figure 1: CSIMS Overview Diagram.

in them is accessible to it (currently not implemented in CSIMS).

- **Information Source Selection:** Given a query, it
  - Determines which classes of information will be relevant to answering the query.
  - Quickly, using some information generated during an earlier preprocessing stage, generates a list of all combinations of sources that contain all information required for a query.
- **Initial Query-Plan Formation:** It creates an initial plan, a sequence of subqueries and other forms of data-manipulation that when executed will yield the desired information. This initial plan does not necessarily satisfy any optimization requirements.
- **Query-Plan Rewriting/Optimization:** By successively applying rewriting rules that preserve the correctness of the plan, it gradually improves the plans efficiency. This process continues until no further rewriting is possibly, or until the allotted time runs out.
- **Execution:** It executes the reformulated query plan; establishing network connections with the appropriate information sources, transmitting queries to them and obtaining the results for further processing. During the execution process CSIMS may detect that certain information sources are not available, or respond erroneously. In such cases, the relevant portion of the query plan will be replanned.

Each information source is accessed through a *wrapper*, a module that can translate from a description of a set of data in CSIMS' internal representation language into a query for that data that is then submitted to the source. The wrapper also handles communication with the information source and takes the data returned by it and sends it on to CSIMS in the form CSIMS expects.

1.2 Information Sources Supported

In order for CSIMS to support an information source it must have a description of the source, and there must exist a wrapper for that type of source. While each information source needs to be described individually, only one wrapper is required for any type of information source.

In addition, through an "ODBC wrapper" CSIMS uses ODBC to interact with all ODBC-enabled databases. This includes Oracle, Sybase, Informix, Ingres, and many others. To add a new database of any of these types requires, therefore, only to create an information source description for it. In order to add an information source of a new type one would have to obtain, or write, a new wrapper for it as well. We also have an ongoing associated effort (Ariadne) that includes work on semi-automatic generation of wrappers for HTML pages.

2 The CSIMS Query Language

Currently, CSIMS only supports commands for retrieving data. Specifically, CSIMS takes a retrieval query as input and returns the data satisfying the constraints specified in the query. The output format of CSIMS is a list of tuples of constant(s). A retrieval query can be expressed in a LOOM syntax or a SQL syntax. The following two sections discuss these two languages in detail.

2.1 LOOM Syntax

Loom serves as the knowledge representation system that CSIMS uses to describe the domain model and the contents of the information sources. In addition, Loom is used to define a knowledge base that itself serves as an information source to CSIMS. Loom provides both a language and an environment for constructing intelligent applications. It combines features of both frame-based and semantic network languages, and provides some reasoning facilities.

The BNF syntax for the CSIMS query language is shown in Figure 2.

```
<query> ::= (sims-retrieve <variable> | ({<variable>}+) <query-expr>)
<query-expr> ::= ({:and | :or} {<query-expr>}+)
<clause> ::= <concept-exp> | <relation-exp> | <assignment-exp> | <comparison-exp>
<concept-exp> ::= (<concept-name> <variable>)
<relation-exp> ::= (<relation-name> {<bound-variable>} {<term>})
<assignment-exp> ::= (:= <unbound-variable> {<arith-exp> | <set-exp>})
<set-exp> ::= ({<constant>}+)
<comparison-exp> ::= <member-comparison> | <arithmetic-comparison>
<member-comparison> ::= (member <bound-variable> <set-exp>))
<arithmetic-comparison> ::= (<comparison-op> {<arith-exp>} {<arith-exp>})
<arith-exp> ::= <number> | <bound-variable> | (<arith-op> <arith-exp> <arith-exp>)
<arith-op> ::= + | - | * | /
<comparison-op> ::= = | > | < | >= | <= | != | match
<concept-name> ::= <symbol>
<relation-name> ::= <symbol>
<term> ::= <constant> | <variable>
<variable> ::= <bound-variable> | <unbound-variable>
<bound-variable> ::= ?<symbol>
<unbound-variable> ::= ?<symbol>
<constant> ::= <number> | <string>
```

Figure 2: BNF for the CSIMS Query Language in LOOM Syntax

The following are the basic forms of a CSIMS query:

```
        (sims-retrieve ?v <query-expr>)
        (sims-retrieve (?v1 ... ?vn) <query-expr>)
```

The variables listed after the sims-retrieve command, $?v$ and $?v_1 \ldots ?v_n$, are considered output variables. This means that the values of these variables are returned as the output of the query. All variables must be named with the prefix '?'. The query expression is composed of clauses and constructors. Clauses determine the values of the variables by binding the variables to specific types of values. In other words, clauses constrain the values of the variables. There are four types of clauses supported by the CSIMS language which will be described in the next section. Clauses can be grouped by constructors into queries. Currently, the constructors provided are :and and :or.

A CSIMS query returns as output a list of instantiations of the output variables which satisfy the bindings of the clauses in the query body. The following shows an example of output from a CSIMS query.

```
    (sims-retrieve (?name)  (:and (American-Large-Seaport ?seaport)
                                  (port-name ?seaport ?name)))
```

```
==> (("Long Beach") ("New York") ("Norfolk") ...)
```

In this query the output variable ?name is bound to the values of the role port-name of American-Large-Seaport.

2.1.1 Clauses

Clauses are expressions that constrain the values which can be bound to a variable. A clause is satisfied when there exists values that satisfy the constraints on the variables in that clause. The following are the four types of clauses:

- Concept expressions:

  ```
  (<concept-name> <variable>)
  ```

  where <concept-name> is the name of a concept, the variable is bound to an instance of the concept <concept-name>. An example of a concept expression is:

  ```
  (Seaport ?seaport)
  ```

  This constrains the variable ?seaport to only the instances of the concept Seaport. Variables in concept expressions cannot be returned by the system.

- Relation expressions:

  ```
  (<relation-name> <bound-variable> <term>)
  ```

  where <relation-name> is the name of a relation, <bound-variable> is a variable from a concept expression while <term> can be either a variable or a constant (a number or a string). The first clause states that there is a binary relation <relation-name> between <bound-variable> and <term>. The following are examples of this type of relation expression:

  ```
  (port-name ?seaport ?name)
  (seaport-country-code ?portCountryCode 'A123)
  ```

  The first expression is only satisfied if the value for ?name is the port-name of ?seaport. The second expression is only satisfied if 'A123 is the seaport-country-code of ?portCountryCode.

- Assignment expressions:

  ```
  (:= <unbound-variable> <arith-expr>)
  ```

  This clause assigns to the unbound variable the computed result of <arith-expr>.

  For the following example, suppose we have a concept Seaport and its relations to its name (port-name) and to its number of cranes code (cranes). The following query will return a list of the names of a pair of seaports that have more than five cranes in total.

  ```
  (sims-retrieve (?portname1 ?portname2)
                 (:and (Seaport ?seaport1)
                       (Seaport ?seaport2)
                       (port-name ?seaport1 ?portname1)
                       (port-name ?seaport2 ?portname2)
                       (cranes ?seaport1 ?cranes1)
                       (cranes ?seaport2 ?cranes2)
                       (:= ?totalcranes (+ ?cranes1 ?cranes2))
                       (> ?totalcranes 5)))
     ==> (("Long Beach" "Norfolk")
          ("New York" "San Diego")
           :
         )
  ```

- Comparison expressions are used to express a constraint on variables. The following are forms of member comparisons:

```
(member <bound-variable> <set-exp>)
```

where a <set-exp> is defined as a set of constants. This clause is satisfied if the variable is bound to one of the constants (i.e., strings or numbers) in the <set-exp>.

The following are examples of member comparisons:

```
(member ?name ("Long Beach" "San Diego" "Newport Beach"))
```

This expression is only satisfied if the value for `?name` matches one of the three strings in the set.

Another type of comparison expression uses the arithmetic comparison operators: `=`, `>`, `<`, `>=`, `<=`, `!=`.

```
(<comparison-op> <arith-expr> <arith-expr>)
```

The following are examples of the arithmetic comparison:

```
(> ?cr 5)
(= ?depth 120)
```

The first example checks that the the number of cranes (?cr) of a seaport is greater than five. The second example verifies the channel depth (?depth) of a seaport is equal to 120.

`Match` is yet another comparison expression. It takes two arguments, a variable and a match string. It matches the value of the variable against the string. The string can have two meta-characters. The first meta-character is %, which matches zero or more characters. The second is _, which matches any one character. The following is an example of using `match`:

```
(sims-retrieve (?name)
   (:and (geographic-location ?g)
         (geographic-name ?g ?name)
         (match ?name "_X%Y%")))
```

This query will retrieve all geographic locations whose names have `X` as the second character and at least one `Y` after the `X`.

2.1.2 Query Expression Constructors

This section describes the two expression constructors supported by CSIMS.

`(:and` *expr*$_1$ ...*expr*$_n$`)` — CONJUNCTION

This returns the values for which each of the expressions *expr*$_j$ is satisfied.

Example: `(:and (Seaport ?x) (port-name ?x ?y))`
This expression is satisfied if `?x` is a `Seaport` and `?y` is the name of that seaport.

`(:or` *expr*$_1$ ...*expr*$_n$`)` — DISJUNCTION

This returns the values for which at least one of the expressions *expr*$_j$ is satisfied.

Example: `(:or (Small-Seaport ?x) (American-Large-Seaport ?x))`
This expression is satisfied if `?x` is either a `Small-Seaport` or an `American-Large-Seaport`.

2.2 SQL Syntax

CSIMS also accommodates queries written in a subset of SQL syntax. A query in SQL syntax is first translated into the native Loom query language and then processed by CSIMS internally. This SQL-syntax front end is different from a typical SQL query engine, such as a relational database, in two important ways.

- Syntax: CSIMS' SQL front end accepts only a subset of standard SQL, a subset which easily corresponds to the internal Loom query language variant used in CSIMS.

- Semantics: CSIMS' SQL front end uses SQL to refer to CSIMS domain concepts and relations, which are high level source-independent descriptions ("views") of the application domain. The terms do not necessarily refer to tables in any particular database.

The BNF syntax for the CSIMS query language is shown in Figure 3.

```
<query> ::= SELECT <ret-param>{,<ret-param>}+
            FROM <concept-spec>{, <concept-spec>}+
            WHERE condition {, condition}+
<ret-param> ::= <colname> | <expr>
<colname> ::= <CONCEPT-NAME>.<ATTRIBUTE-NAME> | <ALIAS>.<ATTRIBUTE-NAME>
<concept-spec> ::= <CONCEPT-NAME> | <CONCEPT-NAME> <ALIAS>
<expr> ::= (<expr> {,expr}+) |
           <constant> |
           -<expr> | +<expr> |
           <expr> <op> <expr>
<constant> ::= <NUMBER> | <STRING> | NULL
<op> ::= * | + | - | /
<comparison ops> ::= = | != | <> | > | < | >= | =< | match
<condition> ::= <expr> <comparison-ops> <expr> |
                <expr> IN <expr> |
                <expr> LIKE <expr> |
                <condition> AND | OR <condition> |
                NOT <condition> |
                (<condition>)
```

Figure 3: BNF for the CSIMS Query Language, SQL Syntax

In both SELECT lists and constraint conditions, attributes must always be specified using the fully qualified (`Concept.attribute`) syntax, even if only a single concept is referenced. This is because parsing of the SQL might take place in an environment where the schema of the underlying view might not be available, so there might be no context providing a way to assign attributes to concepts. The following is a correct example:

```
SELECT ConceptZ.a
FROM ConceptZ
WHERE ConceptZ.b > 10
```

while the next two examples are incorrect because attributes are not specified with the fully qualified syntax.

```
SELECT a
FROM ConceptZ

SELECT ConceptW.b
FROM ConceptW
WHERE b like "%LARGE%"
```

As alluded to above, aliases can be used if desired:

```
SELECT R.a, R.b, S.b, S.c
FROM ConceptX R, ConceptY S
WHERE R.d = S.e
```

However, `Concept.*` (meaning all attributes) is not supported. In addition, CSIMS does not currently distinguish between sets and bags of tuples. Practically, this means that in SELECT statements everything is distinct.

The following is an example of CSIMS query in SQL format

```
SELECT Seaport.port-name
FROM Seaport
WHERE Seaport.cranes > 7
```

```
==> ("Long Beach" "Norfolk" ...)
```

This query asks for the names of large seaports. Equivalently, the following query returns the same information.

```
SELECT Large-Seaport.port-name
FROM Large-Seaport
```

There are a few exceptions in CSIMS' SQL support. Constraints and expressions are currently only expressed in terms of simple arithmetic and boolean operators. In addition, CSIMS' treatment of aggregate operations is currently very limited. Specifically, the following are not supported:

- Nested SELECTs.
- START WITH, GROUP BY, HAVING, CONNECT BY conditions.
- ORDER BY, FOR UPDATE.
- set operations UNION, UNION ALL, INTERSECT, MINUS.
- use of ROWNUM or other pseudocolumns.

Finally, as a (Lisp) syntactic convenience, the system is configured by default to intepret any occurrence of the underscore character (_) in concept names and attribute names to the dash (-) character. For example, the concept-name RUNWAY_LENGTH would be rendered as RUNWAY-LENGTH. This detail is unimportant except for users attempting to match up with a preexisting CSIMS domain model using terms denoted with the (-) character.

3 The Domain Model

A domain model provides the general terminology for a particular application domain. This model is used to unify the various information sources that are available and provide the terminology for accessing those information sources. Throughout this manual we use a simple application domain that involves information about various types of seaports. The example is simple so that we can provide a complete, but short, description of the model. Figure 4 shows our example domain model.

Note: Currently CSIMS does not implement the domain model.

3.1 The Model: Classes and Attributes

The domain model is described in the Loom language, which is a member of the KL-ONE family of KR systems. In Loom, objects in the world are grouped into "classes". Our example domain has several classes: Seaport, Large Seaport, Small Seaport, American Large Seaport, European Large Seaport and Country. The classes are indicated with circles in Figure 4. *Subclass* relationships are shown by dark solid arrows. For example, the class `Large Seaport` is a subclass of `Seaport`. This means that every instance of `Large Seaport` is also an instance of `Seaport`. A class can have any number of subclasses, but (currently) CSIMS allows a class to have at most one superclass.

The figure also shows that `Large Seaport` and `Small Seaport` form a *covering*. This means that the class `Seaport` is the union of `Large Seaport` and `Small Seaport`. Thus, every seaport is either a large seaport or a small seaport.

Classes generally have *attributes* associated with them. For instance, the class `Seaports` has six attributes associated with it, a geographic code (`geoloc-code`), a port name (`port-name`), the number of cranes in the port (`cranes`), the channel depth (`depth`), the seaport's country code (`seaport-country-code`), and a country (`country`). This means that every seaport has a corresponding geographic code, port name, number of cranes, depth, and country code. Attributes are *inherited* down to subclasses. Thus, every large seaport will also have these six attributes, since `Large Seaport` is a subclass of `Seaport`. European large seaports have seven attributes, since they inherit the six attributes from `Large Seaport`, plus there is an additional attribute, the `tariff code`, associated with that class.

Classes can be defined as either primitive classes or they can be defined in terms of other classes. A primitive class has no explicit definition specifying the constraints that differentiate it from its superclass. For example, one might could create the class `Large Seaport` without specifying what constraints differentiate it from its superclass, `Seaport`. In terms of modeling a set of sources, this is useful in the case where you have two sources, where one is clearly a subclass of the other, but there is no simple way to characterize the specific subclass of information it contains.

Alternatively, it is possible to define the relationship between a subclass and superclass by explicitly describing the constraints on the subclass. For example, a large seaport might be defined as a seaport with more than seven cranes. This is what we have done in our example domain, as shown in Figure 4, where the class `Large Seaport` is defined as `Seaport` $\wedge$ `(> cranes 7)`.

3.2 Specifying the Model: Class Definitions

Figure 5 shows the six class definitions that must be given to CSIMS to specify the classes in our example domain. (See Figures 7 and 8 for a BNF description of the modeling language.) The class definition indicates whether or not the class is primitive. When a class is defined in terms of other classes, such as `Large Seaport`, the definition is specified using an "is" clause. The "is" and the "is-primitive" clauses are also used to indicate any attributes associated with the class.

Class definitions also have an "annotations" field. CSIMS requires that every class has at least one defined *key*, which consists of one or more attributes that uniquely identify each instance in a class. Since there may be more than one way to uniquely identify an instance, a class can have multiple keys. For example, any seaport can be uniquely identified either using the `geoloc-code` or the `port-name`. Because more than one attribute may be necessary to uniquely identify an instance, a key can include multiple attributes. For instance, in another domain, it might be that street number, street name and city name are all necessary to uniquely identify a particular house.

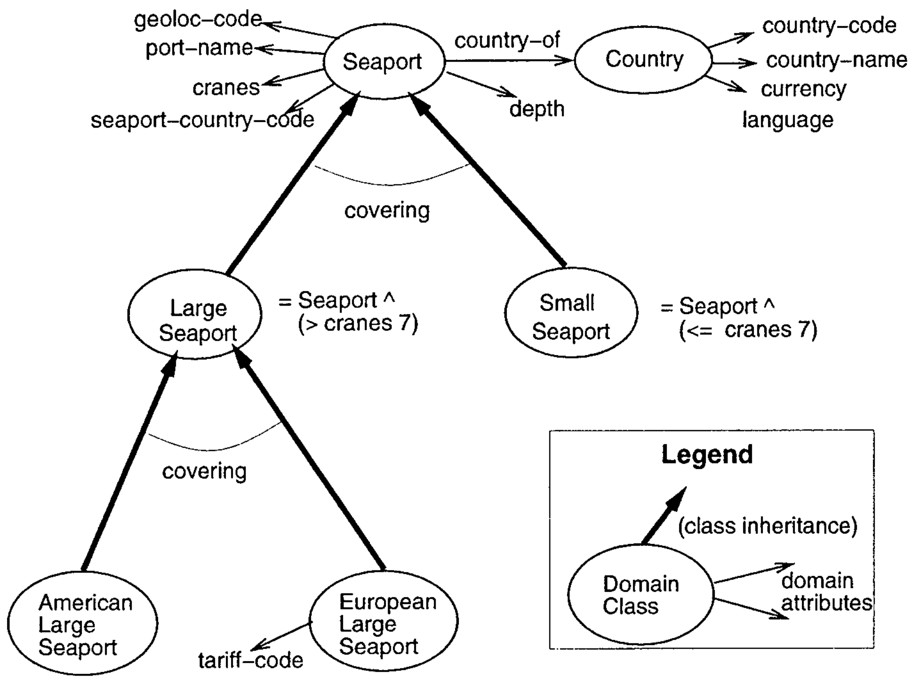


Figure 4: Domain Model

The annotations field of a class definition is also used to indicate that a class is a covering (i.e., the union) of some of its subclasses. For example, the class `Seaport` is the union of `Large Seaport` and `Small Seaport`.

3.3 Specifying the Model: Attribute Definitions

There are two types of attributes in CSIMS. Most of the attributes in our example domain are *simple attributes*, in that they are basic classes: strings or numbers. But attributes can also represent relations between two defined classes. For instance, `Seaport` has an attribute called `Country-of`, so that every seaport is associated with a country. Thus, `Country-of` is a relation between `Seaport` and `Country`.

Figure 6 shows the relation definitions that define the attributes used in the domain model. Notice that each attribute has a domain and range. Defined relations (attributes that relate two classes) have a definition. For instance the `Country-of` relation has a definition which specifies that the relation holds between a seaport and a country if the seaport's `seaport-country-code` matches the country's `country-code`.

There cannot be two different attributes with the same name. In our example, `Seaport` has the attribute `seaport-country-code`, and `Country` has an attribute `country-code`. Even though these are both 'country codes', the names of the attributes must be different.[1]

The domain model is used as the basis for the CSIMS query language that enables the user to construct queries. The classes included in the domain model are not necessarily meant to correspond directly to objects described in any particular information source. The domain model is intended to be a description of the application domain from the point of view of someone who needs to perform real-world tasks in that domain

[1]Had we wanted a `Seaport` to have an attribute called `country-code`, we would have done so only by making the model more complex. For instance, we could have created a class `Geographic entity` with an attribute `country-code`, which could have been a superclass of both `Country` and `Seaport`, in which case the attribute `country-code` would have been inherited down to both of these classes.

```
(def-sims-concept seaport
    :is-primitive (:and sims-domain-concept
                        (:the country-of country)
                        (:the geoloc-code string)
                        (:the seaport-country-code string)
                        (:the port-name string)
                        (:the cranes number)
                        (:the depth number))
    :annotations ((key (geoloc-code))
                  (key (port-name))
                  (covering (large-seaport small-seaport))))

(def-sims-concept large-seaport
    :is (:and seaport
              (> cranes 7))
    :annotations ((key (geoloc-code))
                  (key (port-name))
                  (covering (american-large-seaport
                             european-large-seaport))))

(def-sims-concept small-seaport
    :is (:and seaport
              (<= cranes 7))
    :annotations ((key (geoloc-code))
                  (key (port-name))))

(def-sims-concept american-large-seaport
    :is-primitive large-seaport
    :annotations ((key (geoloc-code))
                  (key (port-name))))

(def-sims-concept european-large-seaport
    :is-primitive (:and large-seaport
                      (:the tariff-code string))
    :annotations ((key (geoloc-code))
                  (key (port-name))))

(def-sims-concept country
  :is-primitive (:and sims-domain-concept
                      (:the country-code string)
                      (:the country-name string)
                      (:the currency string)
                      (:the language string))
  :annotations ((key (country-code))))
```

Figure 5: Class Definitions for our Example Domain

and/or to obtain information about it. CSIMS is designed to allow users to query the domain model without specific knowledge of the way the actual information sources relate to the domain model. The next section describes how application developers describe the actual information sources and their relationship to the domain model.

```
;;; Seaport attributes

(def-sims-relation geoloc-code
    :domain seaport
    :range string)

(def-sims-relation port-name
    :domain seaport
    :range string)

(def-sims-relation cranes
    :domain seaport
    :range number)

(def-sims-relation depth
    :domain seaport
    :range number)

(def-sims-relation seaport-country-code
    :domain seaport
    :range string)

(def-sims-relation country-of
    :domain seaport
    :range country
    :is (:satisfies (?s ?c)
          (:and (seaport ?s)
                (country ?c)
                (seaport-country-code ?s ?cc)
                (country-code ?c ?cc))))


;;; European Large Seaport attributes

(def-sims-relation tariff-code
    :domain european-large-seaport
    :range string)


;;; Country attributes

(def-sims-relation country-code
    :domain country
    :range string)

(def-sims-relation country-name
    :domain country
    :range string)

(def-sims-relation currency
    :domain country
    :range number)

(def-sims-relation lang
    :domain country
    :range number)
```

Figure 6: Attribute Definitions for our Example Domain

```
class-definition ::=
   (DEF-SIMS-CONCEPT ClassName
        is-clause
        annotations-clause)

is-clause ::=
   :IS-PRIMITIVE (:AND SuperClassName attr-clause*)  |
   :IS (:AND SuperClassName constraint-expr*)

annotations-clause ::=
   :ANNOTATIONS (annotation+)

annotation ::=
   (KEY (AttributeName+)) |
   (COVERING (SubClassName SubClassName+))

attr-clause ::=
   (:THE AttributeName ClassName) |
   (:THE AttributeName STRING) |
   (:THE AttributeName NUMBER)

constraint-expr ::=
   (test Term Term) |
   (:FILLED-BY AttributeName Term) |
   (:NOT-FILLED-BY AttributeName Term)

test ::=
   >  |  <  |  >=  |  <=  |  !=  |
```

Figure 7: BNF for Class Definitions

```
simple-relation ::=
  (DEF-SIMS-RELATION RelationName
     :DOMAIN ClassName
     :RANGE [NUMBER | STRING])

defined-relation ::=
  (DEFRELATION RelationName
     :DOMAIN ClassName
     :RANGE ClassName
     :IS (:SATISFIES (VariableName VariableName) constraint-expr))

constraint-expr ::=
   (:FOR-SOME (VariableName) constraint-expr)
   (:AND constraint-expr+) |
   (AttributeName Term Term) |
   (ConceptName Term) |
   (test Term Term)

test ::=
   >  |  <  |  >=  |  <=  |  !=  |
```

Figure 8: BNF for Attribute Definitions

4 Defining Information Sources

In order to extract and integrate data from an information source, a person building an application must describe the contents of the source using terms from the domain model and define the details of how the source is accessed. Each of these issues is addressed in turn.

4.1 Describing the Contents of an Information Source

Each information source is incorporated into CSIMS by describing the data provided by that source in terms of the domain model presented in the previous section. This description provides the following information:

- The precise class of instances provided by a source.
- The set of attributes that are available from the source.
- The name of the source that provides the data (the next section will define additional information about accessing each source)
- The mapping from the table/class name of the source and the name used in the domain model.
- The mapping from the attribute names used in the source and those used in the domain model.

To illustrate the principles involved in representing an information source within CSIMS, consider how a set of sources would be represented using the domain model described in the previous section. Figure 9 shows the example domain model linked to a set of seven separate sources.

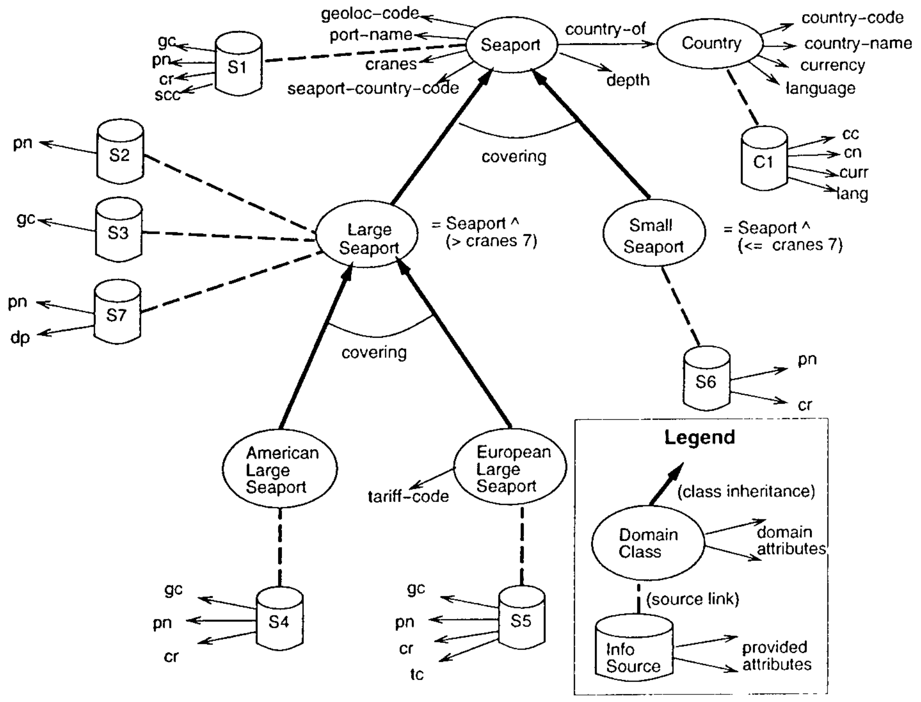


Figure 9: A Set of Sources Described by a Domain Model

In the figure, each source is linked to a class with a dashed line. The meaning of such a link is that the source provides exactly the set of instances described by the class of the domain model. Thus, the figure shows that S2, S3, and S7 all provide *exactly* the same set of large seaports. If there is another source that provides only a subset of a class of instances, then a new subclass in the domain model would be created and the source would be linked to that class. Sources S4 and S5 are both examples of sources that provide subclasses of large seaports and thus are linked to the appropriate subclasses in the domain model.

Since different sources often provide different attributes for the same class, we do not require that all sources provide all attributes of a class. In the figure, the attributes provided by each source are shown next to the individual sources.

The general form of a source description statement is shown in Figure 10. There will generally be one of these statements for each table or relation in a source. However, in some cases, sources can be more naturally modeled by mapping a single relation in the source into more than one domain class. As shown in the figure, a domain class is used to describe a source table and DB and the domain attributes are linked to the corresponding attributes of the source.

```
(source-description <domain-class> <source-table> <source-db>
    (<domain-attribute-1> <source-attribute-1>)
    (<domain-attribute-2> <source-attribute-2>)
    ...
    (<domain-attribute-n> <source-attribute-n>))
```

Figure 10: General Form of a Source Description

Consider how a specific source in Figure 9 would be described. For source S7, which provides the port name and depth of Large-Seaport. The description of this source is shown in Figure 11.

```
(source-description Large-Seaport S7 EXKB7
    (port-name pn)
    (depth dp))
```

Figure 11: Source Description for Large-Seaports table of S7 Database

5 Accessing an Information Source

In addition to specifying the content of an information source, the system also needs to know *what* information sources are currently available and *how* to access them. This section describes the basic commands for declaring information sources.

To make a source available to the system, certain information about the source must be declared in advance. This information is provided in a IS_FILE (information source file). Currently there are four predefined source types, and they are explained in the subsections below.

5.1 kqml-odbc-source

A *kqml-odbc-source* is a source that supports ODBC communication and the full SQL query language. We provide KQML-based wrappers that allow CSIMS to communicate with most of the commericial relational database systems (for details see 8 and 9.1). An example of how such a source would be declared is shown below:

```
(source-type source-name host-name agent-name db-name userid source-status)
```

where:

source-type defines the specific type of source.

source-name provides the name of the information source within CSIMS.

host-name is the name of the machine on which the information source is running.

The *agent-name* of a source is the unique name that KQML uses to identify the wrapper of that particular source.

The *db-name* is the internal database name, which might *not* be unique since one may have multiple instances of the same information source running on different hosts.

The *userid* is the internal `userid` for a database.

source-status defines the status of a source (can be either `UP` or `DOWN`).

Suppose that source S2 from the previous example (Figure 9) is an Oracle database (i.e., it supports ODBC communication) named "assetss" that can be accessed with the userid "abc". It is located on host "isd54.isi.edu" and the wrapper for this source is called "sql_server". We declare this source as follows:

```
(KQML_ODBC_SOURCE "S2" "isd54.isi.edu" "sql_server" "assetss" "abc" UP)
```

5.2 kqml-wrapper-source

A *kqml-wrapper-source* is a wrapper that makes the web sources look like a database and that communicates through KQML interface.

An example of how such a source would be declared is shown below:

```
(source-type source-name host-name agent-name source-status)
```

where:

source-type defines the specific type of source.

source-name provides the name of the information source within CSIMS.

host-name is the name of the machine on which the information source is running.

The *agent-name* of a source is the unique name that KQML uses to identify the wrapper of that particular source.

source-status defines the status of a source (can be either `UP` or `DOWN`).

Suppose that source S3 from the previous example (Figure 9) is a wrapper for a certain website (i.e. www.lg-seaport.com). Since we access this wrapper through KQML, what we need to know are its KQML agent name and the host name where it is located.

```
(KQML_ODBC_SOURCE "S3" "isd54.isi.edu" "lg-seaportwrapper" UP)
```

5.3 http-wrapper-source

A *http-wrapper-source* is a wrapper that runs through CGI. CSIMS communicates with this type of sources via HTTP.

An example of how such a source would be declared is shown below:

```
(source-type domain-name wrapper-name host-name cgi-path port source-status)
```

where:

source-type defines the specific type of source.

domain-name provides the name of the domain name within CSIMS.

wrapper-name provides the name of the wrapper information source within CSIMS.

host-name is the host name of the machine on which the information source is running.

cgi-path is the relative path to where wrapper.cgi resides.

port is the port number that is used for the HTTP server (default : 80).

source-status defines the status of a source (can be either UP or DOWN).

Suppose that source S4 from the previous example (Figure 9) is a http-wrapper for a certain website (i.e. www.lg-seaport.com).

```
(HTTP_WRAPPER_SOURCE "seaport-domain" "S4" "isd56.isi.edu" "/cgi-bin/" 8080 UP)
```

5.4 functional-source

A *functional-source* is a special type of source that is considered to have *in-attributes* and *out-attributes.* Given a set of in-attributes, the functional source provides the corresponding out-attributes. Each functional source is represented as a user defined function that takes as arguments the in-attributes and produces the out-attributes. A user that wants to add a functional source must provide both the source declaration in the IS_FILE and a user-defined function that implements the source. An example of how such a source would be declared is shown below:

```
(source-type source-id line-id source-name source-function source-status)
(source-type source-id line-id In-Attr1 In-Attr2 ...)
(source-type source-id line-id Out-Attr1 Out-Attr2 ...)
```

where:

source-type defines the specific type of source.

source-id is a unique functional source ID.

line-id describes the type of information provided by the current line of the functional source description. It can take the values INFO, IN_ATTR and OUT_ATTR.

source-name provides the name of the information source within CSIMS.

source-function represents the function name of this specific functional source.

source-status defines the status of a source, that can be UP or DOWN.

In the example presented in Figure 9, source "C2" is a functional source that given the country-border length in kilometers provides the corresponding length in miles. If the user-defined function corresponding to this source is called "KmsToMiles", we would declare this source as:

```
(FUNCTIONAL_SOURCE 1 INFO "C2" "KmsToMiles" UP)
(FUNCTIONAL_SOURCE 1 IN_ATTR "km")
(FUNCTIONAL_SOURCE 1 OUT_ATTR "mi")
```

6 The CSIMS Axiom Language

CSIMS uses axioms to determine the set of information sources that can provide the necessary information to answer a specific query. Currently, CSIMS accepts axioms that are provided in the format that we will describe below. Axioms can be provided by RISCSIMS, or they can be written by hand. Given a domain model (see Section 3) and a set of information source definitions (see Section 4), RISCSIMS is a system that produces the relevant axioms for the specified domain. If RISCSIMS is not available, axioms can be written by hand.

6.1 Axiom Syntax

Before defining the actual axioms one has to provide first a *domain schema* and a *source schema.*

6.1.1 Domain Schema

For each concept definition in the domain model (see Figure 5) there is a corresponding definition in the domain schema. In the domain schema, all concept names and attribute names represent domain terms. A concept definition in the domain schema is represented as:

```
domain_concept(domain_attribute-1 domain_attribute-2 ... domain_attribute-n)
```

where:
`<domain_concept> := <string>` , the domain concept name
`<domain_attribute> := <string>` , the domain attribute name
Looking at the class definitions for our example domain (Figure 5) we write the following domain schema.

```
country(country_code country_name currency kilometers language miles)
european_large_seaport(country_of cranes depth geoloc_code port_name seaport_country_code tariff_code)
american_large_seaport(country_of cranes depth geoloc_code port_name seaport_country_code)
large_seaport(country_of cranes depth geoloc_code port_name seaport_country_code)
small_seaport(country_of cranes depth geoloc_code port_name seaport_country_code)
seaport(country_of cranes depth geoloc_code port_name seaport_country_code)
```

6.1.2 Source Schema

For each source in the set of sources described by a domain model (Figure 11) there is a corresponding definition in the source schema. In the source schema, concept names and attribute names represent source terms. A source definition in the source schema is represented as:

```
source_concept(source_attribute-1 source_attribute-2 ... source_attribute-n)
```

where:
`<source_concept> := <source_concept_name>%<table_name>`
`<source_concept_name> := <string>`, the name of the source
`<table_name> := <string>`, the name of the table within the source
`<source_attribute> := <source_attribute_name> | <constant> | <sql_function>`
`<source_attribute_name> := <string> | $<string>`
`<constant> := <string> | <number>`
`<sql_function> := ...`

- String constants are enclosed with double quotes (e.g., "a string").

- Binding pattern annotations ($) must be used when appropriate. A $ in front of an attribute name indicates that the attribute has to be bound in that source.(i.e., it must have a constant value)

For the sources in Figure 11, we write the following source schema:

```
s1%s1(cr gc pn scc)
c1%c1(cc cn curr km lang)
s2%s2(pn)
s3%s3(gc)
s4%s4(cr gc pn)
s5%s5(cr gc pn tc)
s6%s6(cr pn)
s7%s7(dp pn)
c2%c2($km mi)
```

6.1.3 Axiom Definition

Each axiom has a left hand side and a right hand side. The former is expressed in domain terms, while the latter is expressed in source terms. The right hand side may contain a single source definition (*Simple Axiom*), a conjunction of source definitions (*Conjunctive Axiom*), or a disjunction of source definitions (*Disjunctive Axiom*).

```
<axiom> := <simple-axiom> | <conjunctive-axiom> | <disjunctive-axiom>
<simple-axiom> := <domain-concept>({<variable_name>}+) <->
                  <source-concept>({<variable_name>}+)

<conjunctive-axiom> := <simple-axiom> and
                       {<conjunctive-axiom> | <simple-axiom> | <constraint>}

<disjunctive-axiom> := { [<conjunctive-axiom>] | <simple-axiom> }
                       or
                       { [<conjunctive-axiom>] | <simple-axiom> }

<constraint> := <variable> <orderop> <constant>
<variable_name> := ?<string> | _ | ?$<string>
<constant> := <string> | <number>
<orderop> := = | < | > | <= | >=
```

- The number of variables for a domain concept must be identical to the number of domain attributes for that same concept as defined in the domain schema, so that the variable in position "i" maps to the domain attribute in position "i".
- The number of variables of a source concept must be identical to the number of source attributes for that same concept as defined in the source schema, so that the variable in position "i" maps to the source attribute in position "i".
- A $ in front of a variable name indicates that the corresponding attribute must be bound (i.e., it must have a constant value).
- A "_" in the place of a variable name indicates that this axiom does not provide information about the corresponding attribute.

- For <constraints>, if <constant> is numeric, any operator may be used. If <constant> is a string, only "=" is allowed.
- Binding pattern annotations ($) are used in the source clauses (right hand side) and in the domain clauses (left hand side) when the net result of applying all conjuncts would still require a particular attribute to be bound.

Example 1: Given the *domain schema*: `domain_concept(A B C)` and the *source schema*: `source1%source1($A B) source2%source2($B C)` the *axiom* should be:

```
domain(?$A ?B ?C) <->
   source1(?$A ?B) and
   source2(?$B ?C)
```

Note that ?A is marked as a binding variable since no source conjunct establishes it. You should not mark ?B since it is established by source1.

Example 2: If different orders of application can provide distinct axiom binding patterns, multiple axioms, one corresponding to each pattern, should in general be included. Given the *domain schema*: `domain_concept(A B C D)` and the *source schema*: `source1%source1($A B C) source2%source2(A $B D)` the *axioms* should be:

```
domain(?$A ?B ?C ?D) <->
   source1(?$A ?B ?C) and
   source2(?$B ?D)

domain(?A ?$B ?C ?D) <->
   source1(?$A ?B ?) and
   source2(?A ?$B ?D)
```

Simple Axiom Example:

```
seaport(_ ?cranes _ ?geoloc_code ?port_name ?seaport_country_code)
   <-> s1(?cranes ?geoloc_code ?port_name ?seaport_country_code)

small_seaport(_ ?cranes _ _ ?port_name _)
   <-> s6(?cranes ?port_name)

large_seaport(_ _ _ ?geoloc_code _ _)
   <-> s3(?geoloc_code)

country(?country_code ?country_name ?currency ?kilometers ?language _)
   <-> c1(?country_code ?country_name ?currency ?kilometers ?language)
```

Conjunctive Axiom Example:

```
small_seaport(_ ?cranes _ ?geoloc_code ?port_name ?seaport_country_code)
   <-> s1(?cranes ?geoloc_code ?port_name ?seaport_country_code)
       and s6(?cranes ?port_name)

small_seaport(_ ?cranes _ ?geoloc_code ?port_name ?seaport_country_code)
   <-> s1(?cranes ?geoloc_code ?port_name ?seaport_country_code)
       and ?cranes <= 7
```

```
large_seaport(_ ?cranes ?depth ?geoloc_code ?port_name ?seaport_country_code)
   <-> s1(?cranes ?geoloc_code ?port_name ?seaport_country_code)
       and s7(?depth ?port_name)

country(?country_code ?country_name ?currency ?kilometers ?language ?miles)
   <-> c1(?country_code ?country_name ?currency ?kilometers ?language)
       and c2(?kilometers ?miles)
```

Disjunctive Axiom Example:

```
seaport(_ _ _ _ ?port_name _)
   <-> s2(?port_name)
       or s6(_ ?port_name)

seaport(_ ?cranes _ _ ?port_name _)
   <-> s4(?cranes _ ?port_name)
       or s5(?cranes _ ?port_name _)
       or s6(?cranes ?port_name)

large_seaport(_ ?cranes ?depth ?geoloc_code ?port_name _)
   <-> [s4(?cranes ?geoloc_code ?port_name)
        and s7(?depth ?port_name)]
       or [s5(?cranes ?geoloc_code ?port_name _)
           and s7(?depth ?port_name)]
```

6.1.4 SQL Functions

7 The CSIMS Plan Language

We have seen in Section 1.1 that CSIMS can take as input a query, generate a plan for this query, and execute this plan to obtain the query results. In some cases, for instance if we have to execute a query over and over again, it might be desireable to give as an input to CSIMS the *plan* instead of the *query*. In this case, the plan generation code is skipped, and CSIMS just executes the given plan.

The CSIMS plan language is the following:

```
<CSIMS_Plan> := CSIMS_PLAN[ [<initial_node>] [<output_node>]
                {[<retrieve_node>] | {[<optional_node>]}+} ]

<optional_node> := <retrieve_node> | <join_node> | <select_node> |
                   <assignment_node> | <binary_union_node>

<initial_node> := <general_information>

<output_node> := <general_information>

<retrieve_node> := <general_information> Source:<string> SQL:<string>

<join_node> := <general_information> JoinConditions:{<join_expression>+}

<select_node> := <general_information> Selection:<select_expression>
```

```
<assignment_node> := <general_information> AssignmentExpression:<assignment_expression>

<binary_union_node> := <general_information>

<join_expression> := (<orderop> <variable> <variable>)

<select_expression> := (<orderop> <variable> {<variable>|<constant>})

<assignment_expression> := (:= <variable> {<string> | <arith_expression>})

<general_information> := ID:<int> TYPE:<node_type>
                         ProjectionVariables:{<variable>}+
                         From:<int>

<node_type> := InitialNode | output | retrieve |
               join | select | binary_union | assignment

<arith_expression> := <number> | <variable> |
                      (<arithop> <arith_expression> <arith_expression>)

<variable> := ?<string>
<constant> := <string> | <number>
<arithop> := + | - | * | /
<orderop> := = | < | > | <= | >=
```

- The ID of the *initial_node* is always 0.
- The ID of the *output_node* is always 1.
- *From* defines the edges in the plan. For example, "From: 3" denotes that there is an edge in the plan from the node with ID 3 to the current node.
- The *initial_node* always has an empty *From*.
- The *retrieve_nodes* always have "From: 0".
- The *ProjectionVariables* represent the variables that have to be present in the query result of a certain node.
- For a retrieve node we have to specify the SQL query and the source that provides information about this query.

For example, lets consider the query "list the geoloc codes, number of cranes and port name for all large seaports with more than 10 cranes", which has the following Loom representation:

```
(sims-retrieve(?gc ?cr ?pn)
  (:and (large_seaport ?ls)
        (geoloc_code ?ls ?gc)
(cranes ?ls ?cr)
(port_name ?ls ?pn)
(:= ?A 10)
(> ?cr ?A)))
```

and lets consider that the axiom used for this query is:

```
large_seaport(_ ?cranes ?depth ?geoloc_code ?port_name _)
   <-> [s4(?cranes ?geoloc_code ?port_name) and
        s7(?depth ?port_name)]
       or
       [s5(?cranes ?geoloc_code ?port_name _) and
        s7(?depth ?port_name)]
```

We can write the following plan according to the plan language specified above:

```
CSIMS_PLAN[
[
  ID : 0
  TYPE : InitialNode
  ProjectionVariable :
  From :
]

[
  ID : 1
  TYPE : output
  ProjectionVariables : ?geoloc_code0 ?cranes1 ?port_name2
  From : 2
]

[
  ID : 6
  TYPE : retrieve
  ProjectionVariables : ?csx_port_name3
  Source: s7
  SQL : "select distinct large_seaport2.pn from s7 large_seaport2"
  From : 0
]

[
  ID : 7
  TYPE : retrieve
  ProjectionVariables : ?port_name2 ?geoloc_code0 ?cranes1
  Source: s4
  SQL : "select distinct large_seaport1.pn, large_seaport1.gc,
           large_seaport1.cr from s4 large_seaport1"
  From : 0
]

[
  ID : 5
  TYPE : join
  ProjectionVariables : ?geoloc_code0 ?cranes1 ?port_name2
  JoinConditions : (= ?csx_port_name3 ?port_name2)
  From : 6 7
]
[
  ID : 9
  TYPE : retrieve
  ProjectionVariables : ?csx_port_name4
  Source: s7
  SQL : "select distinct large_seaport4.pn from s7 large_seaport4"
  From : 0
]

[
  ID : 10
  TYPE : retrieve
  ProjectionVariables : ?port_name2 ?geoloc_code0 ?cranes1
  Source: s5
  SQL : "select distinct large_seaport3.pn, large_seaport3.gc,
           large_seaport3.cr from s5 large_seaport3"
  From : 0
]

[
  ID : 8
  TYPE : join
  ProjectionVariables : ?geoloc_code0 ?cranes1 ?port_name2
  JoinConditions : (= ?csx_port_name4 ?port_name2)
  From : 9 10
]

[
  ID : 4
  TYPE : binary_union
  ProjectionVariables : ?geoloc_code0 ?cranes1 ?port_name2
  From : 5 8
]

[
  ID : 3
  TYPE : assignment
  ProjectionVariables : ?geoloc_code0 ?cranes1 ?port_name2 ?a
  AxxignmentExpression : (:= ?a 10)
  From : 4
]

[
  ID : 2
  TYPE : select
  ProjectionVariables : ?geoloc_code0 ?cranes1 ?port_name2
  Selection : (> ?cranes1 ?a)
  From : 3
]
]
```

8 Information-Source Wrappers

Once the CSIMS planner has selected the desired sources for a user's query and devised a plan for obtaining the required information, it must communicate with the individual information sources. Sometimes the information source may be complex and difficult to communicate with and additional data processing or functionality may be required. In order to modularize this process and cleanly separate query planning from communication issues, CSIMS requires that for each type of information source there exist a wrapper with which it will communicate. The purpose of the wrapper is to mediate between CSIMS and the information source. The wrapper must be capable of translating between the query language obtained from CSIMS and the information source's query language if necessary, as well as translating between the data output format of the information source and a format appropriate for CSIMS.

This section explains how wrappers are used by CSIMS. The first subsection describes the data that is communicated. The subsequent subsections describe the protocols by which the data is passed; KQML and CORBA.

8.1 Information Source Wrappers

An information source's wrapper will receive a query from CSIMS as input. The syntax of this query language can be varied, so long as the wrapper and CSIMS have agreed upon it. Predefined examples include the CSIMS query language or SQL. See section ?? for more on information sources. One restriction is that all concepts and roles used in the query will be drawn *only* from that information source. Note that at the time when such communication takes place CSIMS has already determined that the query being sent to the information source can be processed in its entirety by that source alone.

The information source's wrapper performs any necessary mediation between CSIMS and the information source. This may involve translating the query into the information source's particular query language, providing additional information to the information source or any other necessary reconciliation between CSIMS and the information source. It then submits a query to the source and retrieves the data. Next, it packages this data into a list of tuples corresponding to the variable parameters used in the submitted query. This tuple is then returned to CSIMS. See Figure 12.

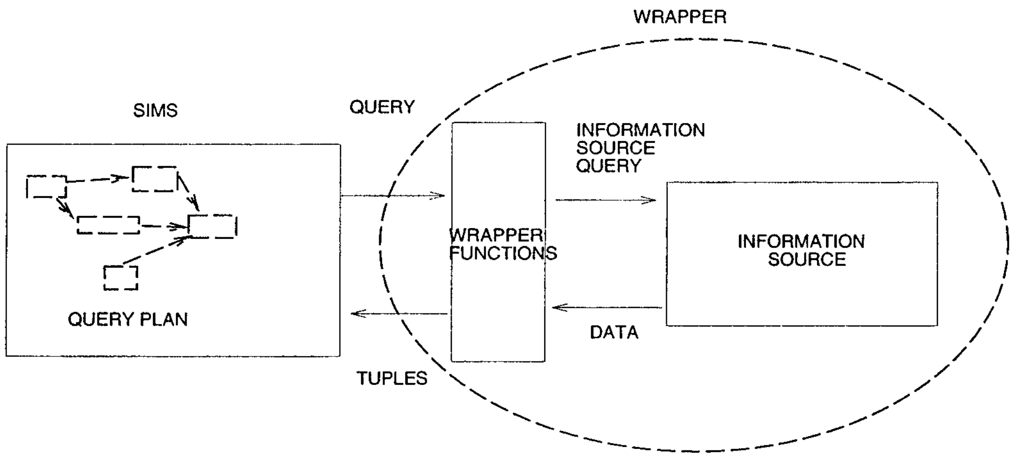


Figure 12: Data Flow between SIMS and Wrappers

In this way, CSIMS is insulated from the particulars of each information source. All the complexity of an information source is hidden from CSIMS via the wrapper module.

9 Communication Issues

9.1 Remote Communication Using KQML

Knowledge Query and Manipulation Language (KQML) protocol, is a language for communication and knowledge sharing between autonomous programs. A simplified view of KQML-based communication is presented in Figure 13.

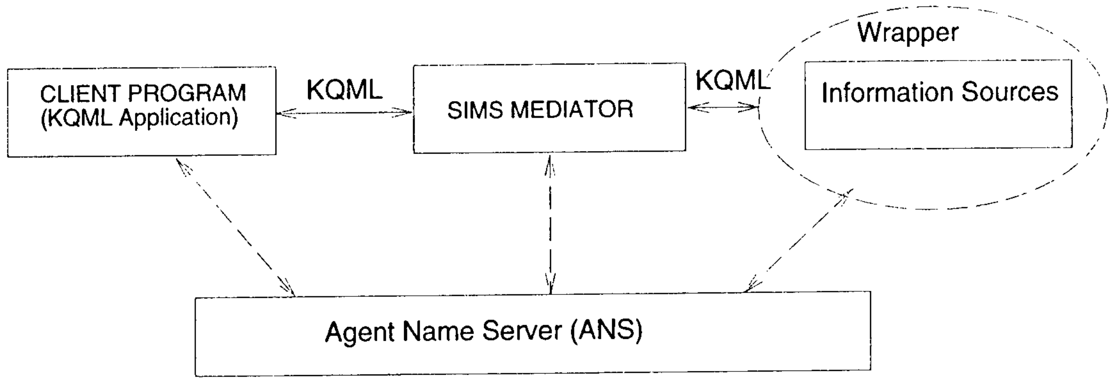


Figure 13: Communication via KQML

For our purpose, KQML provides two main types of functionality that ease the communication between clients and servers (KQML refers to both clients and servers as *agents*). KQML provides a flexible standard language for client-server communication that is available for many platforms, as well as implementation in different languages. It also provides a registry of all clients and servers, so that a client only need to refer to the name registered on the registry by the server (which is usually the name of the service provided and hence more meaningful than just a host address) to communicate with the servers.

The central registry of services in KQML is called the *agent name server (ANS)*, and it records all KQML agents and their addresses. We are mostly interested in the ANS for providing the addresses of information source wrappers CSIMS needs to communicate with (this address resolution process happens transparently and does not require user intervention). The client and server must both be registered with the ANS. The environment variable *KQML_ANS* specifies where the ANS is located, and both the client and server should agree on an ANS accessible to both. An agent is registered using a unique name of the following form, `<user>@<host>-<timestamp>`.

Note that the KQML clients/servers only contact the ANS once to verify the existence of a server and to get its address. The user need not know where a particular server is located but only its name (e.g., SQL-QUERY-SERVER). KQML transparently resolves the location through the ANS and caches each resolved location. The communication protocol used by KQML is TCP/IP. KQML creates a process that listens on a remote TCP/IP stream in order to detect messages from remote hosts.

The ANS used by KQML must be accessible to both CSIMS and to any users of the CSIMS system, but need not be run on those systems itself.

The client must know the messages supported by the server program because only those can be processed. In KQML terms, CSIMS acts as a mediator between the client and the information sources. A KQML mediator receives a request and either delegates it to one or more other servers, or processes it internally/locally (e.g., in a local database). Hence the information source server needs to define a handler for the messages it will support and the client needs to know these messages together with their form.

CSIMS currently uses the *:ASK-ALL* KQML performative to communicate with remote information source servers. The KQML message sent by CSIMS to the information source servers are of the form:

```
(:ASK-ALL :SENDER <SIMS server> :RECEIVER <info-source server>
          :REPLY-WITH T :CONTENT (<SQL query><hostname:dbname><username>))
```

9.2 Remote Communication Using CORBA

As CORBA [1] is supported by virtually all the industry leaders, making CSIMS a CORBA-compliant application broadens the area of potential CSIMS application. For instance, any CORBA-compliant application is able to act like a CSIMS client (i.e., to send queries to CSIMS and to receive the returned answers). A simplified view of CORBA-based communication is presented in Figure 14.

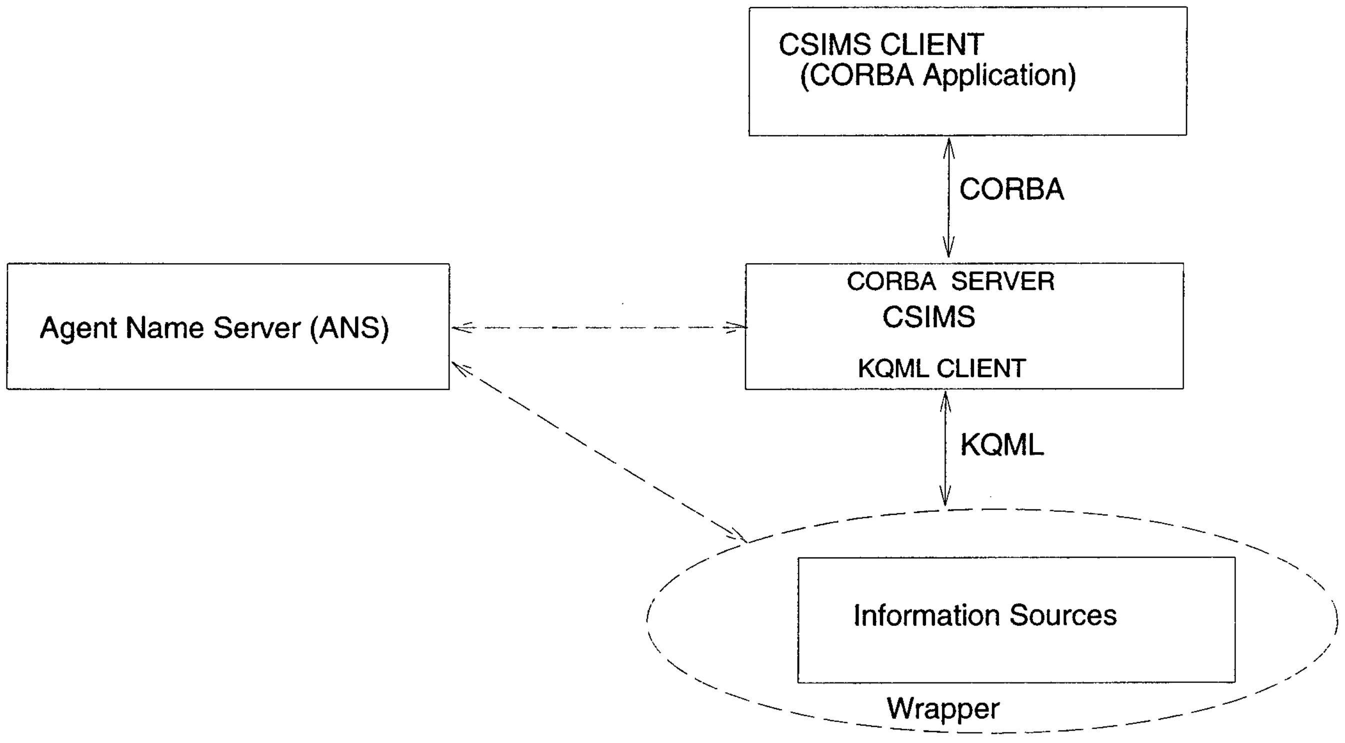


Figure 14: Communication via CORBA

CORBA defines distributed services for inter-process and inter-platform messaging, and it provides interoperability between applications written in different programming languages, running on different machines in heterogeneous, distributed environments. CORBA is an interoperability standard that has several implementations (e.g., Orbix, ILU, VisiBroker), and we currently support the Orbix 2.3 [2] implementation.

Based on our CSIMS-CORBA-Server, any number of CORBA-compliant applications can use CSIMS as a query-answering system.

Figure 15 shows a simple example of a CORBA client that reads a query specified as a command line argument and sends it to the CSIMS server named *CORBA2SIMS* which is located on the machine *vigor.isi.edu*.

```
main(int argc, char **argv)
{
  CORBA2SIMS* anObj = NULL;
  char *query;

  if (argc>=2)  {
    query=argv[1];
  } else {
    printf("Usage: client querystring");
    exit(1);
  }

  TRY{
      anObj = CORBA2SIMS::_bind(":CORBA2SIMS","vigor.isi.edu",IT_X);
  }
  CATCHANY{
      cout << "error" << IT_X << endl;
  }
  ENDTRY

  TRY{
      char * answer=NULL;

      long res = anObj->SendQuery(query, answer);
      if(answer!=0){
cout << answer;
CORBA::string_free(answer);
      }

      if(res!=1){
cout << "SendQuery returned error" << flush;
      }
  }
  CATCH(CORBA::SystemException &se){
      cout << "@@@@ #### CORBA_2_SIMS exception raised!!!" << endl;
      cout << endl << &se <<endl;
  }

}
```

Figure 15: CORBA client

10 Compiling and Running CSIMS

10.1 Compiling CSIMS

In order to compile CSIMS, make the required changes to the `Makefile` (read the file `COMPILE.info` that comes with the distribution), and execute one of the following commands:

- `make planner` : creates the executable `planner`, which is the stand-alone CSIMS;
- `make kqml` : creates the executable `server`, which is the version of CSIMS that can be accessed as a KQML agent;
- `make corba` : creates the executables `CORBA2SIMS` and `corba_client`. The former is the CSIMS CORBA server, while the latter is an example of a CORBA client application.

10.2 Configuring CSIMS

10.2.1 Required Files

CSIMS provides a configuration file that includes all CSIMS configuration variables. Before running CSIMS you must ensure that the system has access to an axiom file (AXIOM_FILE) that contains axioms for a specific domain (in the format specified in Section 6) and a information source file (IS_FILE) that contains the source declarations (see Section 4).

10.2.2 Output Formats

CSIMS can produce the resulted tuples in different formats. The variable OUTPUT_FORMAT specifies the output format. Available formats are: tab delimited format, generic OEM format and dynamic OEM format (the default is dynamic OEM format).

10.2.3 Optimization

By default, CSIMS generates an initial plan for your query, and executes that plan (replanning on failure). If you would like to optimize your initial plan using *Planning by Rewriting (PbR)*, you must set the variable REWRITE, and the variable NR_OF_NEW_PLANS. The higher the number of NR_OF_NEW_PLANS, the better the optimization.

10.3 Using KQML

If KQML is used in the system, set accordingly the KQML_HOME and KQML_HOST variables in the configuration file. KQML_HOST represents the hostname of the KQML ANS, (for details see Section 8).

To start an ANS execute the command:

```
($KQML_HOME)/bin/startans "hostname"
```

In order to check which agents are available in an ANS, or to verify that a service that was registered is up, run the following command in a UNIX shell:

```
($KQML_HOME)/bin/agentls
```

10.4 Using CORBA

If CORBA is used in the system, set accordingly the ORBIX_HOME variable in the configuration file.

To start the CORBA daemon, execute:

```
($ORBIX_HOME)/bin/orbixd
```

In order to check what servers are registered with the daemon, run the command:

```
($ORBIX_HOME)/bin/lsit
```

10.5 Trouble Shooting

In order to be able to handle errors more efficiently, we have created an error hierarchy that describes the types of errors currently handled by CSIMS. These errors are trapped throughout the execution process (by CSIMS itself, by the wrappers, etc) and sent back to the CSIMS server where they are handled appropriately. The current error hierarchy is described in Figure 16.

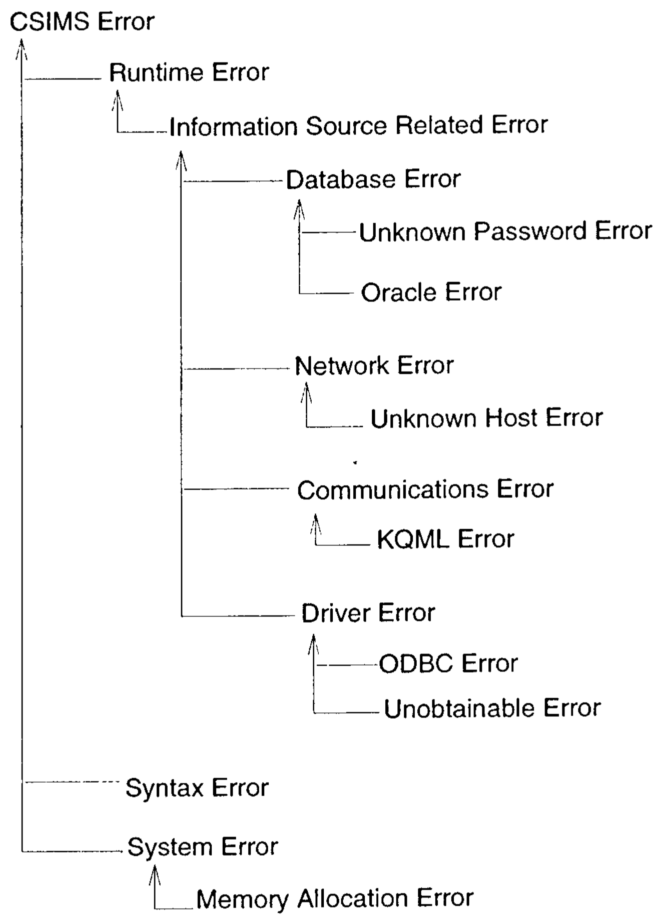


Figure 16: CSIMS Error Hierarchy

In case of an error, CSIMS will display an error message and exit. The error message should provide enough information for you to be able to eliminate the error. Common errors are *Syntax Error* and *Information Source Related Error*. *Syntax Error* occurs if the format of an axiom or the query is incorrect while *Information Source Related Error* can occur during the communicating with the information sources (for example, KQML is down, a database is down, a userid is incorrect, etc.).

CSIMS also provides a tracing mechanism. If the variable CSIMS_TRACE is set (i.e., values 1-7), tracing information is displayed. The higher the value of CSIMS_TRACE, the more information is displayed. The information can be shown on your screen or in a file (the default is on the screen). If the variable TRACE_FILE is set, all trace information is printed to that file.

10.6 Running CSIMS Standalone

To run CSIMS standalone, make the required changes to the configuration file (see Sections 10.2, 10.3, 10.4), execute the configuration file, and then execute the command:

```
planner "query_file"
```

where *planner* is the CSIMS executable, and *query_file* is a file that contains a query in LOOM or SQL syntax.

10.7 Running CSIMS as a KQML Agent

To run CSIMS as a KQML agent, make the required changes to the configuration file (see Sections 10.2, 10.3, 10.4), execute the configuration file, and then execute the command:

```
server "agent_name"
```

where *server* is the KQML agent executable, and *agent_name* is the name by which this agent will be registered with the ANS. The agent expects requests in the form:

```
    (:ASK-ALL :RECEIVER <CSIMS agent>
              :REPLY-WITH T :CONTENT <query>)
```

where "CSIMS agent" is the "agent_name" and "query" is a SQL query.

10.8 Running CSIMS as a CORBA Server

To run CSIMS as a CORBA server, make the required changes to the configuration file (see Sections 10.2, 10.3, 10.4), execute the configuration file, and then register the CSIMS server with the Orbix daemon:

```
putit CORBA2SIMS ($CSIMS_HOME)/CORBA2SIMS
chmodit i+all
chmodit l+all
```

where *CORBA2SIMS* is the CSIMS server executable. Use the generated CORBA client program to test your server. Execute:

```
client "query"
```

where "query" is a SQL or LOOM query.

10.9 Running CSIMS from a CGI Script

CSIMS can also be accessed through HTTP requests that contain the query that CSIMS must execute. The CGI script invokes CSIMS as a stand-alone process (see Section 10.6) and handles the query result received from CSIMS. The CGI script can configure CSIMS according to Section 10.2 by setting the relevant environment variables from the configuration file.

10.10 Running CSIMS from the GUI

First, start the CSIMS server on some port number.

```
planner -port[port_number]
```

You have to have JDK1.2 installed in order to use GUI. Launching a GUI application is as simple as running simsgui.csh. If \$JAVA$_{H}OMEenvironmentvariableisnotset, thesriptwillaskyoutoprovideit.$

```
simsgui.csh
```

Applet version of CSIMS GUI is also available. Use the following template HTML tags to embed a CSIMS GUI applet in an HTML page. Because applet is not allowed to create socket connections to anywhere but the same host from where itself is downloaded, CSIMS server must run on the same host with the HTTP server.

```
<APPLET CODE=SIMS WIDTH=[ number in pixel ] HEIGHT=[ number in pixel ]>
<PARAM NAME=host VALUE="[ host name ]">
<PARAM NAME=port VALUE=[ port number ]>
</APPLET>
```

If your browser supports JDK1.2, you can view the HTML page in the browser, otherwise, use appletviewer that comes with JDK1.2. 'simsguiapplet.csh' is also included in the package.

```
simsguiapplet.csh
```

Users can either type the query (SQL/SIMS) directly to the query fields or select from the avaiable query list in the menu. GUI shows the graph representation of the query plan and also the execution status of the plan. The status of the information sources can also be manipulated through GUI. For detailed instruction on how to use, see `http://ariadne.isi.edu/simsgui.html`.

11 System Requirements

The CSIMS system currently runs in C++ under Solaris on SUN workstations. CSIMS requires the following software components:

KQML (Knowledge Query and Manipulation Language) provides remote communication support between CSIMS and remote DB servers, and between other information integration agents and CSIMS. We are currently using KQML version 2.06. Included with the CSIMS release are several patches which improve the performance of KQML in the CSIMS world. KQML is available by signing a license with University of Maryland, Baltimore County. For more information, see `http://www.cs.umbc.edu/kqml/`.

ODBC. We include with CSIMS code that provides a programmatic interface to Oracle databases using ODBC. We currently use ODBC 3.02 from InterSolv. This component is optional since CSIMS can be run with any database wrapper you may choose to implement.

CORBA. We also include with CSIMS code that allows you to communicate with CSIMS via CORBA. This is optional as well. It does require ORBIX version 2.3 from Iona Technologies to run. For more information on ORBIX, see `http://www.iona.com/`.

STL. CSIMS requires ObjectSpace's `C++ Standard<Toolkit>` version 2.1, which can be obtained from `http://www.objectspace.com/`.

FLEX++ and BISON++. CSIMS requires flex++ version 2.3.8-7 and bison++ version 1.2.1-8 for tokenization and parsing.

12 Coded Example

This section gives the code that implements the example discussed throughout the manual.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; domain-model.lisp
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Domain model for the example in the SIMS user manual
;;;

(in-package :sims)
(in-context :example)

;;; Seaport relations

(def-sims-relation geoloc-code
    :domain seaport
    :range string)

(def-sims-relation port-name
    :domain seaport
    :range string)

(def-sims-relation cranes
    :domain seaport
    :range number)

(def-sims-relation depth
    :domain seaport
    :range number)

(def-sims-relation tariff-code
    :domain european-large-seaport
    :range string)

(def-sims-relation seaport-country-code
    :domain seaport
    :range string)

;;; Seaport concepts

(def-sims-concept american-large-seaport
    :is-primitive large-seaport
    :annotations ((key (geoloc-code))
                  (key (port-name))))

(def-sims-concept european-large-seaport
    :is-primitive (:and large-seaport
                        (:the tariff-code string))
    :annotations ((key (geoloc-code))
                  (key (port-name))))

(def-sims-concept large-seaport
    :is (:and seaport
              (> cranes 7))
    :annotations ((key (geoloc-code))
                  (key (port-name))
                  (covering (american-large-seaport
                             european-large-seaport))))

(def-sims-concept small-seaport
    :is (:and seaport
              (<= cranes 7))
    :annotations ((key (geoloc-code))
                  (key (port-name))))

(def-sims-concept seaport
    :is-primitive (:and sims-domain-concept
                        (:the country-of country)
                        (:the geoloc-code string)
                        (:the seaport-country-code string)
                        (:the port-name string)
                        (:the cranes number)
                        (:the depth number))
    :annotations ((key (geoloc-code))
                  (key (port-name))
                  (covering (large-seaport small-seaport))))

;;; Country relations

(def-sims-relation country-code
    :domain country
    :range string)

(def-sims-relation country-name
    :domain country
    :range string)

(def-sims-relation currency
    :domain country
    :range number)

(def-sims-relation language
    :domain country
    :range number)

;;; Country concepts

(def-sims-concept country
    :is-primitive (:and sims-domain-concept
                        (:the country-code string)
                        (:the country-name string)
                        (:the language string)
                        (:the currency string))
    :annotations ((key (country-code))))

(def-sims-relation country-of
    :domain seaport
    :range country
    :is (:satisfies (?s ?c)
          (:for-some (?country-code)
            (:and (seaport ?s)
                  (country ?c)
                  (seaport-country-code ?s ?country-code)
                  (country-code ?c ?country-code)))))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; queries.lisp
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Example queries
;;;

(in-package :sims)
(in-context :example)

(setq *queries*
  '(;; Large Seaport queries

    (11 "What the geoloc codes of all large seaports"
     (sims-retrieve (?geoloc-code)
       (:and (large-seaport ?ls)
        (geoloc-code ?ls ?geoloc-code))))

    (12 "How many cranes are available in various large seaports"
     (sims-retrieve (?cr)
       (:and (large-seaport ?ls)
        (cranes ?ls ?cr))))

    (13 "List the geoloc-codes and number of cranes for all large
seaports"
     (sims-retrieve (?geoloc-code ?cr)
       (:and (large-seaport ?ls)
        (geoloc-code ?ls ?geoloc-code)
        (cranes ?ls ?cr))))

    ...

    (106 "List currency and language for all countries"
     (sims-retrieve (?cc ?cn ?currency ?lang)
       (:and (country ?c)
        (country-code ?c ?cc)
        (country-name ?c ?cn)
        (currency ?c ?currency)
        (language ?c ?lang))))

    (107 "List all countries' currency, language, and seaport information"
     (sims-retrieve (?cc ?cn ?currency ?lang ?cr ?geoloc-code ?port-name)
       (:and (country ?c)
        (country-code ?c ?cc)
        (country-name ?c ?cn)
        (currency ?c ?currency)
        (language ?c ?lang)
        (seaport ?s)
        (cranes ?s ?cr)
        (geoloc-code ?s ?geoloc-code)
        (port-name ?s ?port-name)
        (country-of ?s ?c))))

    ))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; exkb1.lisp
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Example source model of EXKB1 for SIMS manual
;;;

(in-package :sims)
(in-context :example)

;;; Define a new Loom KB data source called EXKB1

(define-source EXKB1 loom-kb-source)

;;; Describe the domain in terms of EXKB1

(source-description seaport s1 EXKB1
  (geoloc-code gc)
  (port-name pn)
  (cranes cr)
  (seaport-country-code scc))

(source-description country c1 EXKB1
  (country-code cc)
  (country-name cn)
  (language lang)
  (currency curr))

;;; Load data (facts) into this new data source

;;; Seaport data

(deffact EXKB1 S1 ABIDJAN
  (PN "Abidjan")
  (GC "AAPV")
  (CR 5)
  (SCC "IV"))
...
(deffact EXKB1 S1 VLORE
  (PN "Vlore")
  (GC "YALP")
  (CR 1)
  (SCC "AL"))

;;; Country data

(deffact EXKB1 C1 ALBANIA
  (CC "AL")
  (CN "ALBANIA")
  (LANG "ALBANIAN")
  (CURR "LEK"))
...
(deffact EXKB1 C1 ZIMBABWE
  (CC "ZI")
  (CN "ZIMBABWE")
  (LANG "ENGLISH")
  (CURR "DOLLAR"))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; exdb.lisp
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Example source model of EXDB for SIMS manual
;;;

(in-package :sims)
(in-context :example)

;;; Define a new SQL database, addressed using ODBC alias name,
;;; communication via KQML, called EXDB

(define-source EXDB kqml-odbc-sql-source
  :host "isd18.isi.edu"
  :agent-name "sql_server"
  :db-name "examplei"
  :userid "ifd")

;;; Describe the domain in terms of EXDB

(source-description large-seaport lgsp exdb
  (geoloc-code gc)
  (depth dp)
  (port-name pn)
  (cranes cr)
  (seaport-country-code scc))

(source-description small-seaport smsp exdb
  (geoloc-code gc)
  (port-name pn)
  (cranes cr)
  (seaport-country-code scc))

(source-description european-large-seaport lgeurosp exdb
  (geoloc-code gc)
  (depth dp)
  (port-name pn)
  (cranes cr)
  (tariff-code tc)
  (seaport-country-code scc))

(source-description american-large-seaport lgamersp exdb
  (geoloc-code gc)
  (depth dp)
  (port-name pn)
  (cranes cr)
  (seaport-country-code scc))

(source-description seaport sp exdb
  (geoloc-code gc)
  (port-name pn)
  (cranes cr)
  (seaport-country-code scc))

(source-description country ctry exdb
  (country-code cc)
  (country-name cn)
  (language lang)
  (currency curr))

;;; No KB facts
```

13 Additional Reading

Using this manual and following the instructions in it require familiarity with SIMS, as well as with the Loom knowledge representation language, and the KQML transport protocol.

The following papers may be consulted for further information about these programs.

13.1 SIMS

1. Ambite, J.L. and Craig A. Knoblock Reconciling Distributed Information Sources. *Working Notes of the AAAI Spring Symposium on Information Gathering in Distributed Heterogeneous Environments*, Palo Alto, CA, 1995.

2. Ambite, J.L., Yigal Arens, Naveen Ashish, Chin Y. Chee, Chun-Nan Hsu, Craig A. Knoblock Wei-Min Shen, and Sheila Tejada. 1995. *The SIMS Manual*, Version 1.0. ISI/TM-95-428.

3. Arens, Y., Craig A. Knoblock and Chun-Nan Hsu. Query Processing in the SIMS Information Mediator. *Advanced Planning Technology*, editor, Austin Tate, AAAI Press, Menlo Park, CA, 1996.

4. Arens, Y., Chee, C.Y., Hsu, C-N., and Knoblock, C.A. 1993. Retrieving and Integrating Data from Multiple Information Sources. In *International Journal of Intelligent and Cooperative Information Systems*. Vol. 2, No. 2. Pp. 127-158.

5. Arens, Y., Knoblock, C.A., and Shen W-M. Query Reformulation for Dynamic Information Integration, *Journal of Intelligent Information Systems*, 6(2/3):99-130, 1996.

6. Arens, Y. and Knoblock, C.A. 1994. Intelligent Caching: Selecting, Representing, and Reusing Data in an Information Server. In *Proceedings of the Third International Conference on Information and Knowledge Management (CIKM-94)*, Gaithersburg, MD.

7. Arens, Y., Chin Y. Chee, Chun-Nan Hsu, and Craig A. Knoblock Retrieving and Integrating Data from Multiple Information Sources. International Journal of Intelligent and Cooperative Information Systems. Vol. 2, No. 2. Pp. 127-158, 1993.

8. Arens, Y. and Knoblock, C.A. 1992. Planning and Reformulating Queries for Semantically-Modeled Multidatabase Systems, *Proceedings of the First International Conference on Information and Knowledge Management (CIKM-92)*, Baltimore, MD.

9. Hsu, C-N., and Knoblock, C.A. 1995. Estimating the Robustness of Discovered Knowledge, in *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD-95)*, Montreal, Quebec, Canada.

10. Hsu, C-N., and Knoblock, C.A. 1995. Using inductive learning to gen- erate rules for semantic query optimization. In Gregory Piatetsky-Shapiro and Usama Fayyad, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 17. MIT Press.

11. Hsu, C-N., and Knoblock, C.A. 1994. Rule Induction for Semantic Query Optimization, in *Proceedings of the Eleventh International Conference on Machine Learning (ML-95)*, New Brunswick, NJ.

12. Hsu, C-N., and Knoblock, C.A. 1993. Reformulating Query Plans For Multidatabase Systems. In *Proceedings of the Second International Conference of Information and Knowledge Management (CIKM-93)*, Washington, D.C.

13. Hsu, C.-N. and Knoblock, C. A. Discovering Robust Knowledge from Dynamic Closed-World Data. *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, Oregon, 1996.

14. Knoblock, C.A., Arens, Y. and Hsu, C-N. 1994. An Architecture for Information Retrieval Agents. In *Proceedings of the Second International Conference on Cooperative Information Systems*, University of Toronto Publications, Toronto, Ontario, Canada.

15. Knoblock, C.A. 1995. Planning, Executing, Sensing, and Replanning for Information Gathering. In *IJCAI-95*, Montreal, Quebec, Canada.

16. Craig A. Knoblock Applying a General-Purpose Planner to the Problem of Query Access Planning. *Proceedings of the AAAI Fall Symposium on Planning and Learning: On to Real Applications*, 1994.

17. Knoblock, C.A. 1994. Generating Parallel Execution Plans with a Partial-Order Planner. *Artificial Intelligence Planning Systems: Proceedings of the Second International Conference (AIPS94)*, Chicago, IL.

18. Craig A. Knoblock Building a Planner for Information Gathering: A Report from the Trenches Artificial Intelligence Planning Systems: *Proceedings of the Third International Conference* (AIPS96), Edinburgh, Scotland, 1996.

19. Craig A. Knoblock and Jose Luis Ambite. Agents for Information Gathering *Software Agents*, J. Bradshaw ed., AAAI/MIT Press, Menlo Park, CA, 1997.

20. Craig A. Knoblock, Yigal Arens, and Chun-Nan Hsu. Cooperating Agents for Information Retrieval. *Proceedings of the Second International Conference on Cooperative Information Systems*, Toronto, Ontario, Canada, University of Toronto Press, 1994.

These publications, as well as additional information about SIMS, can be accessed through the WWW at `http://www.isi.edu/sims/`.

13.2 Loom

1. MacGregor, R. A Deductive Pattern Matcher. In *Proceedings of AAAI-88, The National Conference on Artificial Intelligence.* St. Paul, MN, August 1988.

2. MacGregor, R. The Evolving Technology of Classification-Based Knowledge Representation Systems. In John Sowa (ed.), *Principles of Semantic Networks: Explorations in the Representation of Knowledge.* Morgan Kaufmann. 1990.

Additional papers and information about Loom can be accessed trough the WWW at the Loom Project homepage: `http://www.isi.edu/isd/LOOM/LOOM-HOME.html` .

13.3 KQML

1. Finin, T., Fritzson, R. and McKay, D. A Language and Protocol to Support Intelligent Agent Interoperability. In *Proceedings of the CE and CALS Washington '92 Conference*, June, 1992.

Additional papers and information about KQML can be accessed through the WWW at the KQML homepage: `http://www.cs.umbc.edu/kqml/` .

13.4 CORBA related

1. Iona Technologies. Orbix 2.2: Programming Guide. March 1997.

2. Object Management group. The Common Object Request Broker: architecture and specification. OMG Document Number 91.12.1, 1991.

Acknowledgements

We would like to thank the developers of the software systems that we have used extensively in the construction of SIMS. In particular, thanks to Bob MacGregor and Tom Russ for the Loom knowledge representation system. Thanks to Don McKay, Jon Pastor, and Robin McEntire at Paramax/Unisys/Loral for their implementation of the KQML language. And thanks to Dan Weld and Tony Barrett at the University of Washington for the UCPOP planner, which we used to build the SIMS planner. In addition, thanks to Ping Luo for his testing of and feedback on an earlier version of this manual.

References

[1] Object Management group. *The Common Object Request Broker: architecture and specification*, volume OMG Document Number 91.12.1. Object Management group, 1991.

[2] Iona Technologies. *Orbix 2.2: Programming Guide.* Iona Technologies, March 1997.